
References


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 distributed algorithm, $P$ a probability distribution, $S$ a schedule, and $\delta \geq 0$. Then $A$ solves a task $T$ with error $\delta$ under schedule $S$ if

$$Pr_P(\{\bar{x} \in X_T | (\bar{x}, output(A, \bar{x}, S)) \not\in T\}) \leq \delta$$

where $output(A, \bar{x}, S)$ is the output of $A$ on input $\bar{x}$ under schedule $S$. Let $\overline{\text{cost}}_\delta(P, S)$, the average cost of task $T$ with error $\delta$ with respect to distribution $P$ and schedule $S$, be the infimum of $\text{cost}(A, P, S)$ taken over all the deterministic algorithms, $A$, that solve $T$ with error $\delta$ under distribution $P$ and schedule $S$.

A randomized distributed algorithm solves a task $T$ with error $\lambda$ under schedule $S$, if for every input $\bar{x}$

$$E((\bar{x}, output(R, \bar{x}, S)) \not\in T) \leq \lambda,$$

where the expectation is over the coin tosses.

A randomized algorithm $R$ solves $T$ with error $\lambda$ if for every $S \in S$, $R$ solves $T$ with error $\lambda$ under $S$. $E_{P, \lambda}$, the randomized cost of task $T$ with error $\lambda$, is the infimum of $E_P(\text{cost}(R))$ taken over all the randomized algorithms that solve $T$ with error $\lambda$.

Using Yao’s result concerning Monte Carlo algorithms, the same technique that was used to prove Theorem 3.1 yields:

**Theorem 5.1** Let $T$ be a distributed task over an 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 $0 \leq \lambda \leq \frac{1}{2}$,

$$\frac{1}{2} \overline{\text{cost}}_{\lambda}(P, S) \leq E_{P, \lambda}.$$

Another result by Yao [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 5.2** Let $T$ be a distributed task over a finite input set $X_T \subseteq X^n$ consisting only of componentwise distinct inputs, $S$ a schedule, and $P$ a probability distribution over $X_T$. Then for every $0 \leq \lambda \leq 1$,

$$(1 - \lambda) \overline{\text{cost}}_{\lambda}(P, S) \leq E_{P, \lambda}/|X_T|.$$

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

**Corollary 5.1** Let $T$ be the task of electing a leader in a synchronous ring of size $n, t$ a positive integer, and $0 < \lambda < 1$. 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 $\lambda/|X_T|$ requires $\Omega(n \log n)$ messages.

6 An Open Problem

We have shown that in the distributed model for every schedule $S$

$$\overline{\text{cost}}(P, S) \leq E_{P}$$

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 $\text{cost}(A(S), P, S) \leq E_{P}$. It remains an open question whether there exists a single deterministic algorithm for which for all schedules $S$ the inequality holds.

For the special case, where the system can be modelled by a single schedule, Corollary 3.1 indeed implies that $\text{cost}(P) \leq E_{P}$. This happens, for example, when modelling 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.
Lemma: 3.3 Let $M > 0$. Let $R$ be a randomized algorithm that solves $T$ with probability 1 and sends no more than $M$ messages, and let $A$ be a canonical representation of $R$. Then with probability 1 an algorithm $A \in A$ solves $T$.

Proof: The bound $M$ on the number of messages allows us to assume that the schedule class $S$ is countable.

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

4 Applications

Like Yao's original method, our results suggest the following technique for proving lower bounds on the randomized complexity of distributed tasks:

1. Find a probability distribution $P$ which gives probability 1 to the set of componentwise distinct inputs, and a schedule $S$ for which a lower bound can be shown on $\text{cost}(P,S)$, the average (with respect to $P$) cost of deterministic distributed algorithms under schedule $S$. (Note that this lower bound has to hold only for deterministic algorithms that are correct for every $S \in S$. This property is important for proving deterministic lower bounds.)

2. Apply Corollary 3.1 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 $\bar{x} \in X_T$ for which every deterministic algorithm satisfies the lower bound — choose a distribution $P$ that gives $\bar{x}$ probability (close to) 1 (and probability (almost) 0 to $X_T - \{\bar{x}\}$).

In 1988 Bodlaender [B] proved an $\Omega(n \log n)$ lower bound on the average message complexity for finding the maximum id in a 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. Doris and Z. Galil [DG] in 1987.

Since Bodlaender’s proof satisfies 1 above, we may use Bodlaender’s result and Corollary 3.1 to show the following lower bound.

Theorem: 4.1 Let $T$ be the task of finding the maximum id in a bidirectional 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 [FL] 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.

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

As a corollary of the theorem we have:

**Corollary 3.1** Let $T$ be a distributed task over an input set $X_T \subseteq X^n$ consisting of only componentwise distinct inputs, $S$ a schedule class, and $P$ a probability distribution over $X_T$. Then for all $S \in S$

$$\overline{\text{cost}}(P, S) \leq E_p.$$

Note that Corollary 3.1 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 3.1 and Corollary 3.1 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. Thus we abandon our methodological assumption that their number is finite.

The only part of the proof that needs an additional effort is in showing that if $A$ is our canonical representation of a randomized algorithm $R$ that is correct with probability 1, then with probability 1, $A \in A$ solves $T$. Since the number of possible schedulers 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 3.2 implies that for every randomized algorithm, $R$, that is correct with probability 1, there exita 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 3.2** Let $R$ be a randomized algorithm that solves $T$ with probability 1. Then for each $\delta > 0$ there is a randomized algorithm $R_\delta$ that solves $T$ with probability 1, such that:

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

(b) $E_p(\text{cost}(R_\delta)) \leq (1 + \delta)E_p(\text{cost}(R))$.

**Proof:** (an outline): For $\vec{x} \in X_T$ and $I \subseteq \{1, \ldots, n\}$, $y^I = (y^I_1, \ldots, y^I_n)$ is a partial output if there exists $y \in Y^n$ for which $(\vec{x}, \vec{y}) \in T$, $y^I_j = y_j$ for $i \in I$ and $y^I_j = -y_j$ otherwise. Let $f$ be a function that maps every input $\vec{x}$ and partial output $y^I$ to a full output $\vec{y'}$ such that $(\vec{x}, \vec{y'}) \in T$ and $\vec{y}$ and $\vec{y}'$ agree on $I$. (If $y^I$ is not a partial output of $\vec{x}$, i.e., there exists no such $y^I$, then $f(\vec{x}, \vec{y'})$ is arbitrary.)

Let $A(\vec{x}, \vec{y'})$ be the algorithm in which every processor $v_i$ broadcasts its private input to all the other processors, and if $i \in I$ it attaches $y_i$ to its private input. Upon receiving the messages from all the processors, $v_i$ computes $\vec{y}' = f(\vec{x}, \vec{y'})$ and outputs $y_i'$. To implement the broadcast, each time a processor gets new information it sends it on all its adjacent vertices. Thus each edge is traversed at most $2(n-1)|E| < n^3$ times.

Given a randomized protocol $R$ and $\delta > 0$, $R_\delta$ is defined as follows: Every processor $v_i$ simulates $R$ until $v_i$ sends $n^3/\delta$ messages and then if $v_i$ did not terminate the protocol, it stops executing $R$ and starts executing $A$ above, with $I$ consisting of all the processors $v_j$ that wrote $y_j$ before switching to $A$. Also, upon receiving a message of protocol $A$, every processor $v_j$ simulates $A$.

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

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

\hspace{1cm} $\Box$
under the uniform distribution on \( \{0,1\}^n \).

The problem with \( R \) is that it does not consist only of uniform algorithms, since for nearly all \( \rho = (\rho_1, \ldots, \rho_n) \in \{\{0,1\}^L\}^n \), \( i \neq j \) implies that \( \rho_i \neq \rho_j \) and therefore \( R[\rho] \), 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.

The following example shows that Yao’s inequality (Lemma 3.1) does not hold in the distributed model.

**The counterexample:** Consider computing the AND function on a ring. In our formulation the private input set is \( X_T = \{0,1\} \). Any deterministic algorithm for this problem has bit complexity \( \Omega(n^2) \) [ASW]. However, by using a randomized algorithm to choose a leader (of a canonical representation — that of uniformity — is violated.)

For each 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 of deterministic algorithms.

The above example demonstrated that in general Yao’s inequality does 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:

**Theorem 3.1** Let \( T \) be a distributed task over an input set \( X_T \subseteq X^n \) consisting of only componentwise distinct inputs, \( S \) a schedule, \( P \) 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

\[
\text{cost}(D, P, S) \leq E_D(\text{cost}(R, S)) \leq E_R(\text{cost}(R)).
\]

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

For each \( \sigma \in \{0,1\}^n \), let \( R_\sigma \) be the deterministic algorithm in which the transition table of each processor is identical to \( \hat{R} \)'s, except that the \( i \)'th coin toss is replaced by the value of \( \sigma_i \) where \( x \) is the private input of the processor.

Let

\[
\mathcal{A} = \{R_\sigma | \sigma \in \{0,1\}^n\}.
\]

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

To show (a), for each \( \sigma \), \( R_\sigma \) is a uniform algorithm since all the processors have the same transition table. Note that the subsequence of \( \rho \) that is used by a processor in \( \hat{R} \), depends only on the processor’s private input and not on its index.

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

\[
\sigma_i (x, i) = \sigma'_i (x, j).
\]

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

For each input \( \bar{x} \) as above, there is a 1-1 correspondence between the above equivalence classes and random inputs \( \rho \in \{\{0,1\}^L\}^n \). If \( \sigma \) belongs to an equivalence class that corresponds to \( \rho \) then the execution of \( R_\sigma \) is equal to that of \( \hat{R} \) with random inputs \( \rho \). Given an equivalence class \( C \subseteq \{0,1\}^n \), 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 \( R_\sigma \in \mathcal{A} \) whose execution is \( \epsilon \) equals to the probability that the execution of the randomized algorithm \( R \) is \( \epsilon \).

We still need to show that each algorithm \( R_\sigma \in \mathcal{A} \) solves \( T \). Since \( R \) is correct for \( T \), for every input \( \bar{x} \) and every schedule \( S \in \mathcal{S} \), every execution of \( R \) on \( \bar{x} \) under \( S \) produces a correct result. Since every execution of \( R_\sigma \) corresponds to some execution of \( R \), we must have that every execution of \( R_\sigma \) must produce a correct result, hence \( R_\sigma \) solves \( T \).
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. Limiting the class of schedules eliminates unwanted behaviors, and may allow efficient algorithms that are correct only for these schedules, thus allowing a smaller cost. Therefore, a lower bound which holds for a restricted class of schedules holds also for a wider class.

Since our algorithms should work for every schedule \( S \) of a schedule class \( \mathcal{S} \), and we are considering worst case behavior, we consider the schedule as an \textit{adversary}, which tries to make the algorithm fail or, at least, consume as much as possible resources.

Some examples of schedules classes are:

1. The synchronous schedule – in step \( i \) all the edges entering processor \( v_i \mod n \) are enabled and all the pending messages (at most 1) in such edges are delivered. (Thus, the computation is conducted in a round-robin fashion, and there is a constant delay of length \( n \) on the edges.)

2. Often an asynchronous system is formalized using the group of all \textit{fair schedules}, i.e., the schedules in which in an infinite execution each edge occurs infinitely often.

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 input buffers are nonempty. In order to simulate wake up messages, we allow a state transition from the initial state even when the buffers are empty. Obviously, any lower bound for a restricted model, also holds for a more general one. In particular, every lower bound on the synchronous model holds for the asynchronous model too.

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 \textit{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 \( \epsilon \), the probability that \( R \) on input \( \vec{x} \) performs execution \( \epsilon \) 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 \( \epsilon \).

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

\textbf{Lemma: 3.1 (Yao)} Let \( T \) be a distributed task over the inputs \( X_T \subseteq X^n \), \( S \) a schedule, \( R \) 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

\[
\sum_{S} P(S) \cdot \sum_{\epsilon} \Pr[R(\vec{x}, S) = \epsilon] = \sum_{S} P(S) \cdot \Pr[A(\vec{x}, S) = \epsilon]
\]

3.2 Main Results

In order to apply Lemma 3.1 above, we need to find canonical representations for distributed randomized algorithms.

First, let us examine the canonical representation of random algorithms that is traditionally used for implementing Yao’s inequality, sometimes implicitly. Recall that \( R[\rho] \) is the deterministic algorithm which results from \( R \) if in the \( i \)th step \( v_j \) uses \( \rho_{j,i} \) (the \( i \)th component of \( \rho_j \)) instead of a coin toss. The traditional technique represents a random algorithm \( R \) by the set

\[
\mathcal{R} = \{ R[\rho] \mid \rho \in \{0,1\}^L \}.
\]
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 $P$, and a schedule $S \in \mathcal{S}$ is:

$$\text{cost}(A, P, S) = \sum_{\bar{x} \in X_T} \text{cost}(A, \bar{x}, S) \cdot Pr_{\bar{x}}(\bar{x}).$$

The average cost of algorithm $A$ with respect to $P$, is the supremum over $\mathcal{S}$ of the average cost of a deterministic algorithm $A$, under schedule $S \in \mathcal{S}$, and is denoted $\text{cost}^{\text{det}}(A, P)$.

The average cost of a task $T$ with respect to $P$, $\text{cost}(P)$, is the infimum of the average cost of $A$ with respect to $P$, taken over all uniform distributed algorithms $A$ that solve $T$.

The average cost of a task $T$, with respect to $P$ and a schedule $S \in \mathcal{S}$ is the infimum of the average cost of $A$ with respect to $P$ and $S$, taken over all uniform distributed algorithms $A$ that solve $T$, and is denoted $\text{cost}(P, S)$. 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 a coin toss. To simplify the notation we assume that the number of coin tosses, performed by 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, $\rho_i$, 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 $\rho = (\rho_1, \ldots, \rho_n) \in \{0, 1\}^n$, the $n$-tuple of private random inputs, is called the random input of the execution.

For a randomized algorithm $R$, let $R(\bar{x}, \rho, S)$ denote the private outputs that result from applying $R$ under schedule $S$ using the random inputs $\rho$ and the input $\bar{x}$. Then $R$ is correct for $T$ if for every $\bar{x} \in X_T$, $\rho \in \{0, 1\}^n$, $S \in \mathcal{S}$,

$$(\bar{x}, R(\bar{x}, \rho, S)) \in T.$$  

The definition of correctness can be weakened to include correctness with probability 1. First we prove our results for the stronger correctness requirement given above, and then generalize it to this latter, weaker definition.

Let $R[\rho]$ be the deterministic algorithm resulting from the randomized algorithm $R$ when processor $v_i$ uses the sequence $\rho_i$ instead of its private random inputs. Since we assume that in each coin toss 0 and 1 are equally likely, each of the $2^n L$ random inputs has equal probability. Therefore, we define the expected randomized cost of algorithm $R$ for input $\bar{x}$ under schedule $S$ to be

$$E_{\rho}(\text{cost}(R, \bar{x}, S)) = 2^{-nL} \sum_{\rho \in \{0, 1\}^n} \text{cost}(R[\rho], \bar{x}, S).$$

The expected randomized cost of algorithm $R$ under schedule $S$ is

$$E_{\rho}(\text{cost}(R, S)) = \sup_{\bar{x} \in X_T} E_{\rho}(\text{cost}(R, \bar{x}, S)),$$

and the expected randomized cost of algorithm $R$ is

$$E_{\rho}(\text{cost}(R)) = \sup_{S \in \mathcal{S}} E_{\rho}(\text{cost}(R, S)).$$

Finally, let $E_{\rho}$, the randomized cost of the task $T$ under schedule class $\mathcal{S}$, be the infimum of $E_{\rho}(\text{cost}(R))$ over all randomized algorithms $R$ that solve $T$. 

3
messages, and an unbounded buffer of messages that have arrived but have not yet been processed by the processor at the receiving end of the channel.

A single transition of processor \( v_i \) consists of receiving (zero or more) messages from some of its incoming channels (i.e., removing the messages from the channels' buffers), 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 messages just received.

Let \( X \) and \( Y \) be the private input set and private output set, respectively. In our distributed model, a processor's actions may depend on its private input \( x \in X \). We also assume that each processor \( v_i \) has a private output, \( y_i \), which can be written only once.

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 (the 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, as will be mentioned later.

A schedule \( S \) is an infinite sequence \( (S_1, S_2, ...) \), where \( S_i = (\langle \epsilon_1, n_1 \rangle, ..., \langle \epsilon_q, n_q \rangle) \) such that \( \epsilon_1, ..., \epsilon_q \in E \) all enter the same vertex \( v \) and \( n_i \in \mathbb{N} \cup \{\infty\} \).

We now describe the \( i \)-th step: Let \( S_i \) and \( v \) be as above. First, for \( k = 1, ..., q \) the first \( n_k \) messages are moved from the queue of pending messages of \( \epsilon_k \) to its buffer. If there are less than \( n_k \) messages (as will always be the case when \( n_k = \infty \)) then all pending messages on \( \epsilon_k \) are moved. Processor \( v \) is then enabled: it receives the messages (i.e., removes them from the buffers) and makes a state transition.

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

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

### 2.2 Distributed Task

A distributed task for \( n \) processors is defined as a relation on \( X^n \times Y^n \). Let \( X_T \subseteq X^n \) be the set of inputs \( \bar{x} \) for which there exists an output \( \bar{y} \in Y^n \) such that \( (\bar{x}, \bar{y}) \in T \). Also, let \( S \) be an arbitrary schedule class.

A distributed algorithm \( A \) is correct for input \( \bar{x} \in X_T \) and schedule \( S \in \mathcal{S} \), if in the execution of \( A \) on \( \bar{x} \) according to \( S \), all processors terminate, and the output \( \bar{y} \) satisfies \( (\bar{x}, \bar{y}) \in T \). A distributed algorithm \( A \) is correct for input \( \bar{x} \in X_T \), if it is correct for every schedule \( S \in \mathcal{S} \). A distributed algorithm \( A \) solves a distributed task \( T \), if \( A \) is correct for every input \( \bar{x} \in X_T \).

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 each \( \bar{x} \in X_T \) has only one component with the value 1, and all the rest 0. The algorithm that writes its private output 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 [Ang].

A cost function is a mapping from the set of all executions to the natural numbers. Given a distributed algorithm \( A \), an input \( \bar{x} \in X_T \), and a schedule \( S \in \mathcal{S} \), let \( \text{cost}(A, \bar{x}, S) \) 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.
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 [Yao]. 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, the inequality may be applied to derive lower bounds on the randomized complexity from known lower bounds on the average complexity.

Yao’s inequality can be immediately applied to additional computational models. For example, the PRAM model (see [FHRW]). However, for the common distributed model, Yao’s technique cannot be applied directly, for two reasons:

1. There is a basic (though somewhat implicit) assumption underlying Yao’s inequality. This assumption is that randomized algorithms can be presented as a probability distribution over a set of deterministic algorithms. It turns out that this assumption depends on the model of computation studied. In particular, we show, by means of a counterexample, that it 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 effect of the existence of a scheduler.

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. The lower bounds we get are either new or improved.

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.

Our techniques, like Yao’s, can be used to deal with Monte Carlo algorithms – algorithms that have some probability of error. By extending Yao’s result, we relate the cost of distributed Monte Carlo algorithms to the average cost of distributed deterministic algorithms which can err on some inputs. In addition, by combining our techniques with another result of Yao, we can show that in some cases the cost of distributed Monte Carlo algorithms, is bounded by the cost of error-free distributed deterministic algorithms.

Independently, Bodlaender [B] proved a result similar to Corollary 3.1 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.

2 Preliminaries

2.1 Distributed Systems

A distributed network of size $n$ consists of a strongly connected directed graph of $n$ vertices, each of which corresponds to a processor. Each processor has unlimited computational power, and its own private memory. Since we are not concerned with computation time, we consider each processor as a (possibly infinite) state machine represented by its transition table. A distributed algorithm is the sequence of the transition tables of the processors. The distributed algorithm is uniform if all the processors have the same transition table. In this case the processors are identical. We are interested in uniform distributed algorithms.

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
Average and Randomized Complexity of Distributed Problems *

Nechama Alanberg-Navony Alon Itai Shlomo Moran

Computer Science Department
Technion, Haifa 32000, Israel

April 26, 1994

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 randomized computations, thus obtaining new results.

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

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