THE ROUND COMPLEXITY OF 1-SOLVABLE TASKS
(Extended Abstract)

by

O. Biran, S. Moran and S. Zaks

Technical Report CS0614

February 1990
THE ROUND COMPLEXITY OF 1-SOLVABLE TASKS
(Extended Abstract)

by

O. Biran, S. Moran and S. Zaks

Technical Report CS0614

February 1990
A distributed task $T$ is $I$-solvable if there exists a protocol that solves it in the presence of (at most) one crash failure. A precise characterization of the $I$-solvable tasks was given in [BMZ]. In this paper we determine the number of rounds of communication that are required, in the worst case, by a protocol which $I$-solves a given $I$-solvable task $T$ for $n$ processors. We define the radius $R(T)$ of $T$, and show that if $R(T)$ is finite, then this number is $\Theta(\log n R(T))$; more precisely, we give a lower bound of $\log (n-1) R(T)$, and an upper bound of $2+\log (n-1) R(T)$. The upper bound implies, for example, that each of the following tasks: renaming, order preserving renaming ([ABDKPRJ] and binary monotone consensus ([BMZ]) can be solved in the presence of one fault in 3 rounds of communications. All previous protocols that $I$-solved these tasks required $\Omega(n)$ rounds. The result is also generalized to tasks whose radii are not bounded, e.g., the approximate consensus and its variants ([DLPSW], [BMZ]).
1. INTRODUCTION

An asynchronous distributed network consists of a set of processors, connected by communication lines, through which they communicate in order to accomplish a certain task; the time delay on the communication lines is finite, but unbounded and unpredictable. In this paper we study the case when at most one processor is faulty, which means that all of its messages are not delivered from some point on (fail-stop failure). It was shown in [FLP] that it is impossible to achieve a distributed consensus for this case. This result was extended in several directions. In [DDS] the features of asynchrony that yield the result of [FLP] and related results were analyzed. In [DLPSW] it was shown that approximate consensus, in which all processors must agree on values that are arbitrarily close to one another, is possible in the presence of few faulty processors. In [ABDKPR] few other problems were shown to be solvable in the presence of faulty processors. In [MW] a class of tasks was shown not to be solvable in the presence of one faulty processor (not 1-solvable). In [BMZ] we provided a complete characterization of the 1-solvable tasks.

Let $T$ be an 1-solvable task. In this paper we analyze the round complexity of $T$, which is the number of communication rounds that are required, in the worst case, by any protocol that 1-solves $T$, and provide optimal bounds (up to a constant factor) for this number. This measure attempts to capture the notion of time complexity for asynchronous, fault tolerant protocols. We first consider bounded tasks, which are tasks that can be 1-solved by protocols that require at most a constant number of rounds in all possible executions (e.g., the renaming tasks and the strong binary monotone consensus task [ABDKPR, BMZ]). Then we generalize our results for unbounded tasks (like the approximate consensus and its variants [DLPSW, BMZ]).

The outline of our proof is as follows: For a distributed task $T$, let $X_T$ be the set of possible input vectors for $T$. First we show, by using the result in [BMZ], that if $T$ is 1-solvable, then there is a set $R_T$ of radius functions related to $T$, where each radius function $p$ is a mapping $p: X_T \rightarrow \mathbb{N}$, which maps each input vector $\mathbf{x}$ on a positive integer $p(\mathbf{x})$. We use this set to define $R(T)$, the radius of the task $T$, as $R(T) = \min_{p \in R_T} \max_{\mathbf{x} \in X_T} p(\mathbf{x})$.

In proving our bounds, we first consider only tasks $T$ for which $R(T)$ is finite, and show that these are exactly the bounded tasks. In fact, we show that if $R(T)$ is finite then the round complexity of $T$ is $\Theta(\log_a R(T))$; more precisely, we give a lower bound of $\log_2(n-1)R(T)$, and an upper bound of $2+\left\lceil \log_2(n-1)R(T) \right\rceil$. We then extend the results to arbitrary task $T$. In the general case, the round complexity of $T$ is not a constant, but a function of the input vector. Since there is no natural total order on these functions, we cannot define the optimal round complexity of $T$, but only define the set of minimal round complexity functions of $T$, in the natural partial ordering of functions. This set is defined by a correspondence to the set of minimal radius functions in $R_T$.

The upper bound implies, for example, that each of the following tasks: renaming with $n+1$ new names, order preserving renaming with $2n-1$ new names ([ABDKPR]), and strong binary monotone consensus ([BMZ]) can be solved in the presence of one fault in three rounds.
of communications. All previous protocols that 1-solved these tasks required $\Omega(n)$ rounds. For the case where $R(T)$ is infinite, we extend the optimal bounds of [Fe] for the approximate consensus. In particular, we show that similar bounds hold for variants of the approximate consensus that were studied in [BMZ], which are considerably harder than the (original) approximate consensus.

The rest of the paper is organized as follows: In Section 2 we provide the preliminary definitions. In Section 3 we define standard protocols and round complexity. In Section 4 we define the radius of a task. The lower and upper bounds for bounded tasks are presented in Sections 5 and 6. In Section 7 we generalize our results for arbitrary tasks and in the Appendix we bring some applications.

2. PRELIMINARY DEFINITIONS AND NOTATIONS

2.1 Asynchronous Systems

An asynchronous distributed system is composed of a set $V = \{P_1, P_2, \ldots, P_n\}$ of $n$ processors $(n \geq 3)$, each having a unique identity, We assume that the identities of the processors are mutually known, and w.l.o.g that the identity of $P_i$ is $i$. Our results are applicable also to the model in which the identities are not mutually known (or absent, provided that the inputs are distinct). The processors are connected by communication links, and they communicate by exchanging messages along them. Messages arrive with no error in a finite but unbounded unpredictable time; however, one of the processors might be faulty, in which case messages might not have these properties (the exact definition is given in the sequel).

2.2 Decision Tasks

Definition: Let $X$ and $D$ be sets of input values and decision values, respectively. A distributed decision task $T$ is a function $$T: X_T \rightarrow 2^D \setminus \{\emptyset\},$$ where $X_T \subsetneq X^n$. $X_T$ is called the input set of the task $T$. The decision set of the task $T$ is the union of the sets $T(x)$ over all $x \in X_T$. Each vector $\vec{x} = (x_1, x_2, \ldots, x_n) \in X_T$ is called an input vector, and it represents the initial assignment of the input value $x_i \in X$ to processor $P_i$, for $i = 1, 2, \ldots, n$. Each vector $\vec{d} = (d_1, d_2, \ldots, d_n) \in D_T$ is called a decision vector, and it represents the assignment of a decision value $d_i \in D$ to processor $P_i$, for $i = 1, 2, \ldots, n$.

Thus, a decision task $T$ maps each input vector to a non-empty set of allowable decision vectors. We assume that all tasks $T$ discussed in this paper are computable, in the sense that the set $(\vec{x}, \vec{d}): \vec{x} \in X_T \text{ and } \vec{d} \in T(\vec{x})$ is recursive.

Examples:

1. Consensus [FLP]: A consensus task is any task $T$ where $X_T = X^n$ for an arbitrary set $X$, and such that $T(\vec{x}) \subsetneq \{(0, 0, \ldots, 0), (1, 1, \ldots, 1)\}$ for every input vector $\vec{x} \in X_T$. Let $\vec{0}$ denote the vector $(0, 0, \ldots, 0)$, and $\vec{1}$ denote the vector $(1, 1, \ldots, 1)$. A strong consensus task is a consensus task $T$, in which there exist two input vectors $\vec{x}$ and $\vec{y}$ such that $T(\vec{x}) = \{\vec{0}\}$ and $T(\vec{y}) = \{\vec{1}\}$. The main result in [FLP]
implies that the strong consensus task is not 1-solvable.

(2) Strong Binary Monotone Consensus [BMZ]: This is probably the strongest variant of the consensus task which is 1-solvable. To simplify the definition, assume that \( n \) is even: The input is an integer vector \( \vec{x} = (x_1, \ldots, x_n) \), and \( T(\vec{x}) \) consists of all vectors \( \vec{d} = (d_1, \ldots, d_n) \) where each \( d_i \) is one of the two medians of the multiset \( \{x_1, \ldots, x_n\} \), and \( d_i \leq d_{i+1} \) (the "strong" stands for the fact that the two values must be the medians).

(3) Renaming [ABDKPR]: This task is defined for a given integer \( K \), where \( K \geq n \). The input set \( X_T \) is the set of all vectors \( (x_1, \ldots, x_n) \) of distinct integers. For a given input \( \vec{x} \), \( T(\vec{x}) \) is the set of all integer vectors \( \vec{d} = (d_1, \ldots, d_n) \) satisfying \( 1 \leq d_i \leq K \) and such that for each \( i, j, d_i \neq d_j \). In order to prevent trivial solutions in which \( P_i \) always decides on \( i \), this task assumes a model in which the processors identities are not known.

(4) Order Preserving Renaming (OPR) [ABDKPR]: This task is similar to the renaming task, with the additional requirement that for each \( i, j, x_i < x_j \) implies \( d_i < d_j \).

(5) Approximate Consensus [DLPSW]: This task is defined for any given \( \epsilon > 0 \). The input set \( X_T \) is \( \mathbb{Q}^n \), where \( \mathbb{Q} \) is the set of rational numbers, and for a given input \( \vec{x} = (x_1, \ldots, x_n) \), \( T(\vec{x}) \) is the set of all vectors \( \vec{d} = (d_1, \ldots, d_n) \) satisfying \( |d_i - d_j| \leq \epsilon \) and \( m \leq d_i \leq M \) (\( 1 \leq i, j \leq n \)), where \( m = \min(x_1, \ldots, x_n) \) and \( M = \max(x_1, \ldots, x_n) \).

(6) Strong Binary Monotone Approximate Consensus [BMZ]: This is a harder variant of the approximate consensus task which is still 1-solvable. To simplify the definition, assume that \( n \) is even: The input is the same as for the approximate consensus. For an input \( \vec{x} = (x_1, \ldots, x_n) \), \( T(\vec{x}) \) consists of all vectors \( \vec{d} = (d_1, \ldots, d_n) \) satisfying \( \vec{d} \) has at most two distinct entries, which lie between the two medians of the multiset \( \{x_1, \ldots, x_n\} \), and \( d_i \leq d_{i+1} \leq d_{i+2} \).

2.3. Protocols and Executions

A protocol for a given network is a set of \( n \) programs, each associated with a single processor in the network. Each such program contains operations of sending a message to a neighbor, receiving a message and processing information in the local memory.

If the network is initialized with the input vector \( \vec{x} \in X_r \) (i.e., the value \( x_i \) is assigned to processor \( P_i \)) and if each processor executes its own program in the protocol \( \alpha \), then the sequence of operations performed by the processors is called an execution of \( \alpha \) on input \( \vec{x} \). (We assume that no two operations occur simultaneously; otherwise, we order them arbitrarily. For more formal definitions see, e.g., [KMN].)

Definition: A vector \( \vec{d} = (d_1, d_2, \ldots, d_n) \) is an output (decision) vector of \( \alpha \) on input \( \vec{x} \) if there is an execution of \( \alpha \) on \( \vec{x} \) in which processor \( P_i \) decides on \( d_i \), by writing \( d_i \) in a write-once register.

2.4. Faults and 1-Solvability

Definition: A processor \( P \) is faulty in an execution \( e \) if all the messages sent by \( P \) during \( e \) from some point on are never received (a fail-stop failure; see, e.g., [FLP]. Also known as crash failure; see, e.g., [NT]).

Definition: A protocol \( \alpha \) 1-solves a task \( T \) if for every execution of \( \alpha \) on input \( \vec{x} \in X_T \) in which at most one processor is faulty, the following two conditions hold:

(1) All the non-faulty processors eventually decide.
(2) If no processor is faulty in the execution, then the output vector belongs to $T(G)$. 

When such a protocol exists, we say that the task $T$ is 1-solvable.

The definition above does not require the processors to halt after reaching a decision. However, in the case of a single failure, it is not hard to see that a processor that learns that $n-1$ processors had already decided may halt. Hence, in this case, reaching a decision by all non-faulty processors is sufficient to guarantee halting. For this reason, in this paper we shall restrict the discussion to protocols in which the processors are guaranteed to halt in every possible execution. (Note that in the case of $k > 1$ crash failures, there exist tasks which can be $k$-solved only by protocols that do not guarantee termination, e.g., the renaming tasks [ABDKPR]. For more on the termination requirement for multiple failures see [TKM]).

3. STANDARD PROTOCOLS AND ROUND COMPLEXITY

In this paper we bound the number of communication rounds that are required by protocols that 1-solve a given task. This number attempts to capture the notion of time complexity for asynchronous, fault tolerant protocols, in which every processor is guaranteed to halt. We model an arbitrary $t$-resilient protocol that works in rounds of communications by the notion of standard protocol. The definitions and discussion below are restricted to the case $t=1$.

3.1. Standard Protocols

A protocol that 1-solves a task $T$ is standard protocol if it works in rounds of communications, as follows. In each round a processor broadcasts a message (which includes the round number), which is a function of its state, to all the processors (including itself), and waits until it receives $n-1$ messages of this round (or less if it heard on processors that had already halted). During this period of waiting, it might receive messages from different rounds. Those of higher rounds are saved until the processor itself reaches these rounds. Messages of previous rounds (might be one such message per each previous round) - called late messages - are gathered with the $k-1$ of this round to form a set $M$. Then the processor computes its next state, which is a function of $M$ and its previous state. The state of a processor includes its write-once register.

Our notion of standard protocol is similar to the one used in [Fe]. It can be shown that this notion is general enough for the sake of lower bounds, by using full information protocols [Fe, FL].

Formally, the standard protocol for $P_k$:

$\begin{align*}
  r &\leftarrow 0 \\
  \text{state} &\leftarrow \text{INIT}_k \\
  \text{while} \ \text{state} < \text{HALT} \ \text{do} \\
  &\begin{align*}
  r &\leftarrow r+1 \\
  \text{broadcast} \ (r, \ \text{MESSAGE\_FUNCTION}_k(\text{state})) \\
  \text{wait until you receive} \ n-1 \ \text{messages of form} \ (r, \ *) \\
  M &\leftarrow \\text{late message} \ (r, \ M) \\
  \text{state} &\leftarrow \text{STATE\_FUNCTION}_k(\text{state}, \ M) \\
  \end{align*}
  \end{align*}$
3.2. Round Complexity

Definition: Let $T$ be a task and $\alpha$ a standard protocol that $1$-solves $T$. The \textit{round complexity of $\alpha$ on input $\vec{x}$}, denoted $rc_{\alpha}(\vec{x})$, is the maximum round number, over all executions of $\alpha$ on input $\vec{x}$, that a correct processor reaches.

The \textit{round complexity of $\alpha$}, denoted $rc_{\alpha}(T)$ is defined by: $rc_{\alpha}(T) = \max_{\vec{x} \in \mathcal{X}_T} rc_{\alpha}(\vec{x})$.

The round complexity $rc(T)$ of a task $T$ is defined by: $rc(T) = \min \{ rc_{\alpha}(T) \mid \alpha \text{ $1$-solves } T \}$.

Note that $rc(T)$ may be infinite; this is the case only when the input set $\mathcal{X}_T$ is infinite, and for any protocol $\alpha$ that $1$-solves $T$ and for any constant $C$, there is an input $\vec{x}$ such that $rc_{\alpha}(\vec{x}) > C$.

Definition: A $1$-solvable task $T$ is \textit{bounded} iff $rc(T)$ is finite, and is \textit{unbounded} otherwise.

We will first present results for bounded tasks, and then extend them to results which are applicable for unbounded tasks as well.

4. COVERING FUNCTIONS AND RADII OF TASKS

We first give some basic definitions from [BMZ] which are needed for this paper.

4.1 Adjacency graphs, partial vectors, covering vectors and $i$-anchors

Definition: Let $S \subseteq \mathbb{A}^n$, for a given set $A$. Two vectors $\vec{s}_1, \vec{s}_2 \in S$ are \textit{adjacent} if they differ in exactly one entry. The \textit{adjacency graph} of $S$, $G(S) = (S, E_s)$, is an undirected graph, where $(\vec{s}_1, \vec{s}_2) \in E_s$ iff $\vec{s}_1$ and $\vec{s}_2$ are adjacent. For a task $T$ and an input vector $\vec{x}$ for $T$, $G(T(\vec{x}))$ is the \textit{decision graph} of $\vec{x}$.

Definition: A \textit{partial vector} is a vector in which one of the entries is not specified; this entry is denoted by $\star$. For a vector $\vec{s} = (s_1, \cdots, s_n)$, $\vec{s}^i$ denotes the partial vector obtained by assigning $\star$ to the $i$-th entry of $\vec{s}$, i.e., $\vec{s}^i = (s_1, \cdots, s_{i-1}, \star, s_{i+1}, \cdots, s_n)$. $\vec{s}$ is called an \textit{extension} of $\vec{s}^i$.

Definition: Let $\vec{x}^i$ be a partial input vector and $\vec{d}^i$ a partial decision vector of a task $T$. We say that $\vec{d}^i$ is a \textit{covering vector} for $\vec{x}^i$ if for each extension of $\vec{x}^i$ to an input vector $\vec{x} \in \mathcal{X}_T$, there is an extension of $\vec{d}^i$ to a decision vector $\vec{d} \in T(\vec{x})$.

Definition: A vector $\vec{d}$ is an \textit{$i$-anchor} of an input vector $\vec{x}$ if $\vec{d} \in T(\vec{x})$ and $\vec{d}^i$ is a covering vector for $\vec{x}^i$.

Example: consider the OPR task for $n=3$ processors and $K=5$. For the partial input vector $\vec{x}^3 = (10, \star, 30)$ there is a unique covering vector $\vec{d}^3 = (2, \star, 4)$, and the input vector $\vec{x} = (10, 20, 30)$ has a unique 2-anchor $\vec{d} = (2, 3, 4)$. In the OPR task with $n=3$ and $K=6$ there are three covering vectors for $\vec{x}^2$: $(2, \star, 4)$, $(2, \star, 5)$, and $(2, \star, 5)$. Thus, $\vec{x}$ has four 2-anchors: $(2, 3, 4)$, $(2, 3, 5)$, $(2, 4, 5)$, and $(3, 4, 5)$.\[^{+1}\]
4.2 Covering functions and radii of tasks

Definition: A covering function for a given task $T$ is a function that maps each partial input vector to a corresponding covering vector for it.

Definition: Let $T$ be a task, $CF$ a covering function for $T$, and $\vec{x} \in X_T$ an input vector. An anchors tree for $\vec{x}$ based on $CF$ is a tree in $G(T(\vec{x}))$ that, for each $i$ ($1 \leq i \leq n$), includes an $i$-anchor which is an extension of $CF(\vec{x}^i)$.

We now reformulate Theorem 3 of [BMZ] to a form suitable to our discussion:

Theorem [BMZ]: A task $T$ is 1-solvable if and only if there exists a covering function $CF$ for $T$, s.t. for each input vector $\vec{x} \in X_T$, there is an anchors tree for $\vec{x}$ based on $CF$. □

A covering function satisfying the condition of Theorem [BMZ] is termed a solving covering function for $T$.

We are now ready to define the concept of the radius of a task $T$, which plays an essential role in this paper. All the definitions below refer to an arbitrary 1-solvable task $T$.

Definition: Let $CF$ be a solving covering function for $T$, and $\vec{x}$ an input vector in $X_T$. $\rho_{CF}(\vec{x})$ is the minimum possible radius of an anchors tree for $\vec{x}$ based on $CF$.

By the above definition, each solving covering function $CF$ defines a radius function $\rho_{CF}: X_T \rightarrow N$. The set of all radius functions for $T$ is denoted by $R_T$. That is,

$$R_T = \{\rho_{CF} : CF \text{ is a solving covering function for } T\}.$$

$R(T)$, the radius of the task $T$, is defined by:

$$R(T) = \min_{\rho_{CF} \in R_T} \max_{\vec{x} \in X_T} \rho_{CF}(\vec{x}).$$

Note that $R(T)$ may be infinite; this is the case only when the input set $X_T$ is infinite, and for any radius function $\rho_{CF}$ in $R_T$ and for any constant $C$, there is an input $\vec{x}$ such that $\rho_{CF}(\vec{x}) > C$.

As we shall show, $R(T)$ is finite if $T$ is a bounded task.

A covering function $CF$, and the corresponding radius function $\rho_{CF}$, are optimal for a bounded task $T$ if $\max_{\vec{x} \in X_T} \rho_{CF}(\vec{x}) = R(T)$.

Example: Consider the following task $T$, in which $X_T$ composed of only 3 input vectors:

$\vec{x}_1 = (50,20,30)$, $\vec{x}_2 = (10,20,30)$ and $\vec{x}_3 = (10,20,70)$.

$T(\vec{x}_1) = \{(5,2,3)\}$,

$T(\vec{x}_2) = \{(1,2,3),(1,4,3),(5,4,3),(5,4,6),(7,4,6),(7,5,6),(7,5,8),(3,5,8),(3,2,8)\}$ and

$T(\vec{x}_3) = \{(7,4,1),(3,2,1)\}$.

Now, in choosing an optimal covering function for $T$, the only partial input vectors that should be considered are those which might be extended to more than one input vector (if $\vec{x}^i$ might be extended to a unique input vector $\vec{x}$, then any vector $\vec{d}$ in $T(\vec{x})$ is an $i$-anchor of $\vec{x}$, so the need to select an $i$-anchor does not impose any constrain on the anchors tree). Thus we consider only $(*,20,30)$ and $(10,20,*)$, so the only anchors that constrain the anchors tree are the 1-anchor and
the 3-anchor. From the decision graphs (see Figure 1), clearly \( X_2 \) is dominate for \( R(T) \), since any anchors tree of the others two is compose of a single vertex. There are only 2 covering functions (which are different in the 2 key partial vectors):
\[ CF_1((*,20,30)) = (*,2,3), CF_1((10,20,*)) = (7,4,*) \] and
\[ CF_2((*,20,30)) = (*,2,3), CF_2((10,20,*)) = (3,2,*). \]
In the anchors tree based on \( CF_1 \) (in \( G(G_2) \)) the 1-anchor is \((1,2,3)\), the 3-anchor is \((7,4,6)\), and thus the radius is 2 (a line, with center \((5,4,3)\) ). In anchors tree based on \( CF_2 \) the 3-anchor is \((3,2,8)\), and the radius is 4. So \( CF_1 \) is the optimal covering function, and \( R(T) = 2 \).

More examples appear in the Appendix.

5. LOWER BOUND

The following proposition shows that for proving lower bounds, it suffices to consider standard protocols that do not use late messages, since the use of late messages cannot improve the round complexity.

**Proposition 1:** Let \( \alpha \) be a standard protocol which I-solves a task \( T \). Then the protocol \( \alpha' \), which is identical to \( \alpha \) except it doesn't use late messages (i.e., the set \( M \) in line 7 of the standard protocol is built only of round \( r \) messages - the late messages are ignored) also I-solves \( T \), and for every input \( \vec{x} \), \( r_{\alpha'}(\vec{x}) \leq r_{\alpha}(\vec{x}) \).

**Proof:** For each execution (which is determined by the messages scheduling) of \( \alpha' \) on input \( \vec{x} \) consider a similar execution, in which all late messages are delivered only after \( r_{\alpha}(\vec{x}) \) rounds. \( \alpha' \) clearly works the same in both executions (namely, the states of the processors after each round are the same). But up to round \( r_{\alpha}(\vec{x}) \) in the second execution, \( \alpha' \) works exactly as \( \alpha \), and thus is guaranteed to halt with legal output vector before round \( r_{\alpha}(\vec{x}) + 1 \).

**Theorem 1:** Let \( T \) be a bounded task. Then its round complexity \( r_{c}(T) \) satisfies
\[ r_{c}(T) \geq \log_{n}(n-1)^{R(T)} \]

**Proof:** We first give an upper bound for the number of output vectors that might be reached by a protocol \( \alpha \) with \( r_{c}(T) = k \) in executions on one input vector.

For the following lemmas we need the definition below:

**Definition:** For a protocol \( \alpha \) and an input vector \( \vec{x} \), \( D_{\alpha}(\vec{x}) \) is the set of all possible decision vectors of \( \alpha \) on input \( \vec{x} \).

**Lemma 1:** Let \( \alpha \) be a standard protocol that does not use late messages, which I-solves a task \( T \). If \( r_{c}(T) = k \) then for each input vector \( \vec{x} \), \( |D_{\alpha}(\vec{x})| \leq n(n-1)^{k} \).

**Proof:** An execution of \( \alpha \) on input vector \( \vec{x} \) is completely determined by the messages that each processor receives, round after round. Since a processor always receives its own message, there are at most \( n-1 \) possibilities for each processor in a round (it may be less if some processors halt before the current round). Thus the number of executions on \( \vec{x} \) is bounded by \( n(n-1)^{k} \), and this is clearly a bound for \( |D_{\alpha}(\vec{x})| \).
For Lemma 2, which completes the proof of Theorem 1, we need a definition and a former result:

**Definition:** Let $\alpha$ be a protocol that 1-solves a task $T$. An *i-sleeping execution* of $\alpha$ is an execution in which all the messages sent by $P_i$ are delayed until all other processors decide (such an execution exists by the definition of 1-solvability, since $P_i$ is not distinguishable from a faulty processor).

The following is Corollary 1 in [BMZ]:

**Proposition [BMZ]:** Let $\alpha$ be a protocol which 1-solves a task $T$. Then, for each input vector $\vec{x} \in X_T$, the graph $G(D_{\alpha}(\vec{x}))$ is connected.

**Lemma 2:** Let $\alpha$ be a protocol that 1-solves a task $T$, and assume there exists an integer $q$ such that for each input vector $\vec{x}$, $|D_{\alpha}(\vec{x})| \leq q$. Then $R(T) \leq q$.

**Proof:** Define a solving covering function $CF_{\alpha}$: for each partial input vector $\vec{x}^i$ define $CF_{\alpha}(\vec{x}^i)$ to be a partial vector output by the $n-1$ processors (without $P_i$) in an i-sleeping execution on $\vec{x}$ (it's not hard to see that this is indeed a covering vector, since $\alpha$ 1-solves $T$). For each input vector $\vec{x}$ define $A_i$ an i-anchor of $\vec{x}$ (i.e., $i \leq n$), to be the output vector reached when continuing the i-sleeping execution on $\vec{x}^i$ (that output $CF_{\alpha}(\vec{x}^i)$) by waking $P_i$ with input that extends $\vec{x}^i$ to $\vec{x}$.

Now, since for each $\vec{x}$, $G(D_{\alpha}(\vec{x}))$ is connected (Corollary 1 [BMZ]) and $|D_{\alpha}(\vec{x})| \leq q$, it follows that there is a tree with radius $q$ or less in $G(D_{\alpha}(\vec{x}))$ which includes the $n$ $A_i$'s. In particular, $\max_{\vec{x} \in X_T} \rho_{CF_{\alpha}}(\vec{x}) \leq q$. The Lemma follows by the fact that $\rho_{CF_{\alpha}} \in R_T$. \(\square\)

### 6. UPPER BOUND

#### 6.1. The protocol

**Theorem 2:** The round complexity of a bounded task $T$ is at most $2 + \lceil \log(n-1) \rceil R(T)$.

**Proof:** We present a protocol that 1-solves $T$, and whose round complexity is $2 + \lceil \log(n-1) \rceil R(T)$. The protocol is an improvement of the protocol in [BMZ], whose round complexity is $2 + R(T) (2+2R(T)$ if the number of processors, $n$, is 3). Like the protocol in [BMZ], this protocol is based on a given solving covering function $CF$. Informally, this protocol differs from the one in [BMZ] in two ways. First, in each execution of this protocol all the vectors that may be suggested by the processors belong to a single path in the anchors tree of $CF$, while the protocol in [BMZ] may use a larger portion of that tree. Second, the convergence to two adjacent vertices on that path is done by an averaging process, similar to the one used in approximate consensus protocols, and not in the step by step fashion of the protocol in [BMZ].

Let $CF$ be an optimal solving covering function of $T$ (i.e., $R(T) = \max_{\vec{x} \in X_T} \rho_{CF}(\vec{x})$). By the computability of $T$, it follows that there is an algorithm $TREE$ that on input $\vec{x}$ outputs a minimum radius anchors tree $TREE(\vec{x})$ based on $CF$, with a center $ROOT(\vec{x})$ as its root. Our
The general outline of the algorithm is as follows: In the first two stages each processor $P_k$ is trying to find out the input vector $\vec{x}$. For this, it first broadcasts its input value and receives $n-1$ input values (including its own), which determine a partial input vector $\vec{x}^j$ (note that $j \neq k$). Then it broadcasts $\vec{x}^j$ and waits for $n-1$ such partial vectors. At this point, there are two kinds of processors: those who know only partial input vector $\vec{x}^j$ (it is the same $\vec{x}^j$ for all these processors), and those who know the complete input vector $\vec{x}$.

Now, the processors perform a simple averaging approximate consensus, for $\lceil \log_{n-1} R(T) \rceil$ rounds, with two kinds of initial values: those who know $\vec{x}^j$ start with zero, and those who know $\vec{y}$ start with $R(T)$. During these rounds, each of the processors appends to its messages whatever it knows from the two things - $\vec{x}$ and $\vec{y}$. After these rounds, each processor will have a value $v$ in $[0, R(T))$ s.t. the maximum difference between the values is 1. If $v$ is equal to zero (in this case $P_k$ still knows only $\vec{x}^j$) then $P_k$ decides on $CF(\vec{x}^j)$ (deciding on a (partial) output vector $(d_1, \ldots, d_k, \ldots, d_r)$ means, in particular, that $d_k$ is the decision value of $P_k$). Otherwise $P_k$ knows $\vec{x}$ (and thus can compute $TREE(\vec{x})$; actually, it will only have to compute $ROOT(\vec{x})$, or the path in $TREE(\vec{x})$ from the j-anchor to $ROOT(\vec{x})$). If $v$ is equal to $R(T)$, then $P_k$ decides on $ROOT(\vec{x})$. Otherwise, $P_k$ knows $\vec{x}$ and $\vec{y}$. Then, it normalizes the value $v$ to the length of the path from the j-anchor to $ROOT(\vec{x})$ (which is less or equal $R(T)$), and decides on the $p$-th (the normalized value) vector on this path. Since the difference between the $v$ values was at most 1, this ensures that each non-faulty processor will decide on one out of two adjacent vertices (vectors) (this guarantees that the actual output vector is one of these two vectors, and hence it is in $T(\vec{x})$).

The protocol for $P_k$:

A. **BROADCAST** $\vec{x}_k$ and **WAIT** until you RECEIVE $n-1$ stage-A messages

B. you know $\vec{x}^j$, **BROADCAST** $\vec{x}^j$ and **WAIT** until you RECEIVE $n-1$ stage-B messages

C. For $r = 1$ to $\lceil \log_{n-1} R(T) \rceil$ do
   
   - if you know only $\vec{x}^j$ then $v \leftarrow 0$
   - else if $v = R(T)$ you know only $\vec{y}$ then **DECEIDE** $ROOT(\vec{x})$
   - else (you know $\vec{x}$ and $\vec{y}$) do
     
     - **BROADCAST** $(r, info, v)$ and **WAIT** until you RECEIVE $n-1$ messages of round $r$
     - $v \leftarrow$ the average of the $n-1$ $v$'s received in this round
   
   end

D. if $v = 0$ (you know only $\vec{x}^j$) then **DECEIDE** $CF(\vec{x}^j)$
   else if $v = R(T)$ (you know only $\vec{y}$) then **DECEIDE** $ROOT(\vec{x})$
   else (you know $\vec{x}$ and $\vec{y}$) do
     
     - Let $l$ be the length of the path in $TREE(\vec{x})$ between the j-anchor and $ROOT(\vec{x})$
     - $p \leftarrow \lfloor v / R(T) \rfloor$
     - **DECEIDE** on the $p$ 'th vector of the path in $TREE(\vec{x})$ between the j-anchor and $ROOT(\vec{x})$ (the j-anchor is number 0 in the path, and $ROOT(\vec{x})$ number 1)
   
   end

**HALT**
6.2 Correctness proof

In the correctness proof we will show that all non-faulty processors decide on two adjacent vectors in the path between a \( j \)-anchor to \( \text{ROOT}(\vec{x}) \). This \( j \) is determined if some processors know only \( \vec{x}^j \) after stage B (such \( \vec{x}^j \) is unique since \( n-1 \) is a majority). If there are no such processors, then they all decide on \( \text{ROOT}(\vec{x}) \).

After \( \lceil \log_{(n-1)} R(T) \rceil \) rounds of approximate consensus in stage C the difference between the \( v \)'s will be at most 1 (since it is reduced at least by a factor of \( n-1 \) each round). If no processor finish it with \( v = R(T) \) or \( v = 0 \) then clearly the maximum difference between the \( p \)'s is 1 since \( l \leq R(T) \). If some processor finished with \( v = R(T) \) (and decided on \( \text{ROOT}(\vec{x}) \)) then all the \( v \)'s are in the range \( (R(T)-i, R(T)) \), and the minimum possible \( p \) is \( l-1 \) (the number of the vector adjust to \( \text{ROOT}(\vec{x}) \)). The argument for \( v = 0 \) is similar (deciding on \( CF(\vec{x}^j) \) is exactly like deciding on the \( j \)-anchor). \( \Box \)

7. GENERALIZATION

In this section we generalize our results to hold for arbitrary tasks. In the general case, the round complexity of a protocol that 1-solves (possibly unbounded) task \( T \) is not a constant, but a function from the set of input vectors \( X_T \) to the positive integers.

**Definition:** Let \( T \) be a 1-solvable task. A function \( f : X_T \rightarrow \mathbb{N} \) is a round complexity function of \( T \) if there exists a protocol \( \alpha \) that 1-solves \( T \), and for each \( \vec{x} \in X_T \) \( r_{\text{CF}}(\vec{x}) \leq f(\vec{x}) \) (\( r_{\text{CF}}(\vec{x}) \) is defined in Section 3.2).

Since in general there is no natural total order on such functions, we cannot define the optimal round complexity of a task \( T \), but only define the set of minimal round complexity functions of \( T \), in the natural partial ordering of functions, as follows.

**Definition:** Let \( f \) and \( g \) be two functions defined on the same domain \( X \). Then \( f \) is smaller than \( g \) if \( \forall x \in X, f(x) \leq g(x) \). A function \( g \) is minimal in a set of function \( F \) if there is no \( f \in \mathcal{F} \) such that \( f \) is smaller than \( g \).

We define the set of minimal round complexity functions of a task \( T \) by a correspondence to the set of minimal radius functions in \( \mathcal{R}_T \): we show that for each minimal radius function \( \rho_{\text{CF}} \) in \( \mathcal{R}_T \) there corresponds a minimal round complexity function which is \( \Theta(\log_{(n-1)} \rho_{\text{CF}}) \), and these are the only minimal round complexity functions of \( T \).

**Theorem 1:** Let \( T \) be a task and \( \rho_{\text{CF}} \) be a minimal radius function in \( \mathcal{R}_T \). Then, there is no round complexity function of \( T \) which is smaller than \( \log_{(n-1)} \rho_{\text{CF}} \).

**Proof:** Similar to the proof of Theorem 1, and using the minimality of \( \rho_{\text{CF}} \). \( \Box \)

We now extend the upper bound to hold also for unbounded 1-solvable tasks. In fact, the next theorem show that for each radius function \( \rho_{\text{CF}} \) in \( \mathcal{R}_T \) there is a protocol whose round complexity for each input \( \vec{x} \) is \( O(\log_{(n-1)} \rho_{\text{CF}}(\vec{x})) \).
**Theorem 2u:** Let $\rho_{CF}$ be a radius function for a task $T$. Then, $3 + \lceil \log_{\alpha-1} \rho_{CF} \rceil$ is a round complexity function of $T$.

**Proof:** We only need few minor changes in the protocol of Section 6: First, all occurrences of $R(T)$ are replaced by $\rho_{CF}(T)$. Now, the problem is that processors that at the beginning of stage C know only $z^j$, cannot compute $\lceil \log_{\alpha-1} \rho_{CF}(T) \rceil$ - the number of approximate consensus rounds. To solve this problem, we add an initialization round in stage C (this idea is borrowed from [DLPSW]) in which a processor that receives a message with $v = 0$ sets its own $v$ to 0, and a processor that all the $n-1$ values it receives are 0 (and thus still knows only $z^j$), broadcasts a "FINISH" message, and exits stage C. A processor that receives in the next rounds a "FINISH" message, sets its $v$ to 0, broadcasts a "FINISH" message and exits stage C. Thus, if some processor broadcasts "FINISH" message in the initialization round, then all processors set their $v$ to 0, and it follows that all the $v$'s will be zero after stage C. The rest of the correctness proof is similar to the one in Section 6. □

**REFERENCES**


Figure 1: A task $T$ with $R(T) = 2 (= \rho_{CF_1}(\mathcal{X}_2))$.
APPENDIX: APPLICATIONS

We present here new optimal bounds on the round complexity of the 1-solvable tasks mentioned in the paper. The first three examples deal with bounded tasks, and provide upper bounds of 3 rounds for the tasks involved (it can be shown that 2 rounds are not enough). All previous protocols that 1-solved these tasks required \( \Omega(n) \) rounds. The bounds are proved by presenting a covering function \( CF \) for each task \( T \) which prove that \( R(T) \leq n-1 \) (and hence \( \log n R(T) \leq 1 \)). Actually, each of the covering functions presented will be optimal. The last example deal with the strong binary monotone approximate consensus, and provide a bound of approximately the same bound that is proved optimal in [Fe] for the task of approximate consensus, which seems to be considerably simpler than the strong binary monotone approximate consensus. (We note, however, that the bounds in [Fe] apply to multiple failures.)

The formal definitions of the tasks discussed below are given in Section 2.2.

1. **Binary Monotone Consensus**: Let \( \mathcal{X}^i = (x_1, \ldots, x_{i-1}, *, x_{i+1}, \ldots, x_n) \) be a partial input vector for this task. Again, we assume for simplicity that \( n \) is even. In this case there is a unique possible covering function \( CF \), defined by \( CF(\mathcal{X}^i) = (c, \ldots, c) \), where \( c \) is the median of the multiset \( \{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\} \).

We now describe anchors trees based on \( CF \). For a given input vector \( \mathcal{X} \), let \( c \) and \( d \) be the two medians of the multiset \( \{x_1, \ldots, x_n\} \). If \( c = d \) then the anchors tree consists of the single vertex \( (c, \ldots, c) \). Otherwise, it consists of the path \( \{(c, \ldots, c, d), (c, \ldots, c, d, d), \ldots, (c, d, \ldots, d)\} \). In the first case the radius of the tree is 0, and in the second is \( \frac{n}{2} - 1 \).

It can be shown that this anchor tree is of minimum possible radius, and hence \( R(T) = \frac{n}{2} - 1 \).

2. **Renaming with \( n+1 \) new names**: In this task the input to each processor is its id, and the id's are not mutually known. Such a task cannot be modeled as a function from input vectors to output vectors, since there is no fixed order among the processes. Instead, it is modeled as a function between input sets to allowed output sets [BMZ]. By adapting the definitions for this model, as done in [BMZ], we get a that \( R(T) \leq n-1 \).

3. **Order Preserving Renaming with \( 2n-1 \) new names**: This task is order invariant, i.e: \( T(\mathcal{X}) \) depends only on the relative order among the entries of \( \mathcal{X} \). \( CF \) is also order invariant, and we describe \( CF(\mathcal{X}^i) \) only for the case that the entries in \( \mathcal{X} \) are monotone increasing (i.e., \( x_i < x_{i+1} \)). The adaptation of the definition to other order types is straightforward. In this case, \( CF(\mathcal{X}^i) = (2, 4, \ldots, 2i-2, *, 2i, \ldots, 2n-2) \). A suitable anchors tree of such \( \mathcal{X} \) is the path of length \( 2n - 2 \) (and hence of radius \( n-1 \)) starting at \((1, 2, 4, \ldots, 2n-2)\) and ending in \((2, 4, \ldots, 2n-2, 2n-1)\), that passes via all the \( i \)-anchors. (e.g., for \( n = 3 \) this path is \([1, 2, 4, 3, 2, 4, 3, 2, 4, 3] \)).
(4) Binary Monotone Approximate Consensus (for a given \( \epsilon \)): The input is the same as for the binary monotone consensus. The (unique) covering function \( CF \) is the same as for the binary monotone consensus. The minimal radius anchors tree based on \( CF \) is also similar to the one for the binary consensus, but this time \( \epsilon \) must be taken into account:

For a given input vector \( \overrightarrow{x} \), let \( c \) and \( d \) be the two medians of the multiset \( \{x_1, \ldots, x_n\} \). Assume for simplicity the \( \epsilon \) divides \( d - c \). If \( c = d \) then the anchors tree consists of the single vertex \((c, \ldots, c)\). Otherwise, it consists of the path \((c, \ldots, c, c+\epsilon), (c, \ldots, c+\epsilon, c+\epsilon), \ldots, (d-\epsilon, \ldots, d-\epsilon, d-\epsilon), \ldots, (d, \ldots, d-\epsilon)\). In the first case the radius of the tree is 0, and in the second is \( n \left( \frac{d-c}{\epsilon} \right) \). Thus, the upper bound provided by our results (for unbounded tasks) is at most \( 5 + \log_{1-\epsilon} \left( \frac{d-c}{\epsilon} \right) \).