TIGHT BOUNDS ON THE ROUND COMPARE OF DISTHBUTED 1-SOLVABLE TASKS*

Ofer Biran
IBM Haifa Research Group
Technion city, Haifa Israel 32000

Shlomo Moran and Shmuel Zaks
Department of Computer Science
Technion, Haifa, Israel 32000

December 3, 1991

Abstract

A distributed task T is 1-solvable if there exists a protocol that solves it in the presence of (at most) one crash failure. A precise characterization of the 1-solvable asks 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 1-solves a given 1-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 $\Theta(log_n R(T))$;

*This research was supported in pwt by Technic V.P.R. Funds – Wellner Research Fund and Loewengart Research Fund, and by the Foundation for Research in Eletronics, Computers and Communications, administrated by the Israel Academy of Sciences and Humanities, and by the ESPRIT II Basic Research Actions Program of the EC under contract no. 3075 (project ALCOM). A preliminary version of this paper appeared in the proceedings of the 4th International Workshop on Distributed Algorithms, Bari, Italy 1990.

more precisely, we give a lower bound of $\log_{(n-1)} R(T)$, and an upper bound of $2 + \lceil \log_{(n-1)} R(T) \rceil$. The upper bound implies, for example, that each of the following tasks: renaming, order preserving renaming ([ABDKPR]) and binary monotone consensus ([BMZ]) can be solved in the presence of one fault in 3 rounds of communications. All previous protocols that 1-solved these tasks required $\Omega(n)$ rounds. The result is also generalized to tasks whose radii are not funded, 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 unpredictible. 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 [DLS] the features of asynchrony that yield the result of [FLP] and related results were analyzed. The possibility of reaching agreement when restricting the pure asynchrony was studied also in [ADG, DLS]. 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] the solvability of two renaming problem (which will be defined later) in the presence of faults was investigated. In [MW] a class of tasks was shown not to be solvable in the presence of one fault processor (not 1-solvable). In [BMZ] we provided a complete characterization of the 1-solvable tasks.

In this paper we are interested in the *round complexity* of a 1-solvable task, which is the number of communication rounds that are required, in the worst case, by any protocol that 1-solves it. us measure attempts to capture the notion of *time complexity* for asynchronous, fault tolerant protocols. In [Fe], a tight bound was given for the specific task of the approximate consensus. Result of the same type in other models were given in [ALS] for the approximate consensus task in asynchronous shared memory, and in [HT] for me renaming task in synchronous message passing model.

We provide optimal bounds (up to an *additive* constant) on the round complexity of a general 1-solvable task. 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, hen there is a set $\mathbf{R}_T$ of *radius functions* related to T, where each radius function ρ is a mapping $\rho : X_T \rightarrow N$, which maps each input vector $\vec{x}$ to a positive integer $\rho(\vec{x})$. We use this set to define $R(T)$, the radius of the task T, as

$$R(T) = \min_{\rho \in \mathbf{R}_T} \sup_{\vec{x} \in X_T} \rho(\vec{x}).$$

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. We show that if $R(T)$ is finite then the round complexity of T is $\Theta(log_n R(T))$; more precisely, we give a lower bound of $\log_{(n-1)} R(T)$, and an upper bound of $2 + \lceil \log_{(n-1)} R(T) \rceil$. 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 $\mathbf{R}_T$.

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: 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 bided tasks are presented in Sections 5 and 6. In Section 7 we generalize our results for arbitrary tasks and in Section 8 we present some applications.

2 PRELIMINARY DEFINITIONS AND NOTATIONS

2.1 Asynchronous Systems

An *asynchronous distributed system* is composed of a set $P = \{P_1, P_2, \cdots, P_n\}$ 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. it 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 and 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 me 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

$$T : X_T \rightarrow 2^{D^n} - \{\emptyset\},$$

where $X_T \subseteq X^n$. X_T is called the *input set* of the task T. D_T, the *decision set* of the task T, is the union of the sets $T(\vec{x})$ over all $\vec{x} \in X_T$. Each vector $\vec{x} = (x_1, x_2, \cdots, x_n) \in X_T$ is called an *input vector*, and it represents the initial assignment of the *input value* $x_i \in X$ to processor P_i, for $i = 1, 2, \cdots, n$. Each vector $\vec{d} = (d_1, d_2, \cdots, d_n) \in D_T$ is called a *decision vector*,

and it represents the assignment of a *decision value* $d_i \in D$ to processor P_i, for $i = 1, 2, \cdots, n$.

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 $T(\vec{x}) \subseteq \{0, (0, \cdots, 0), (1, 1, \cdots, 1)\}$ for every input vector $\vec{x} \in X_T$. Let $\vec{0}$ denote the vector $(0, 0, \cdots, 0)$, and $\vec{1}$ denote the vector $(1, 1, \cdots, 1)$. A *strong* consensus task is a consens task T, in which there exist two input vectors $\vec{u}$ and $\vec{v}$ such that $T(\vec{u}) = \{\vec{0}\}$ and $T(\vec{v}) = \{\vec{1}\}$. The main result in [FLP] implies that a 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 $\vec{x} = (x_1, \cdots, x_n)$, and $T(\vec{x})$ consists of all vectors $d = (d_1, \cdots, d_n)$ where each d_i is one of the two medians of the multiset $\{x_1, \cdots, x_n\}$, and $d_i \leq d_{i+1}$ (the "strong" stands for the fact that the two values must be the medians).

3. **Renaming** [ABDKPR]: his task is defined for a given integer K, where $K \geq n$. The input set X_T is the set of all vectors $(x_1, \cdots, x_n)$ of distinct integers. For a given input $\vec{x}$, $T(\vec{x})$ is the set of all integer vectors $\vec{d} = (d_1, \cdots, d_n)$ 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, is task assumes a model in which the processors identities are not known.

4. **Order Preserving Renaming (OPR)** [ABDKPR]: This task is similar to be 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 $\varepsilon > 0$. The input set X_T is Q^n, where Q is the set of rational

numbers, and for a given input $\vec{x} = (x_1, \cdots, x_n)$, $T(\vec{x})$ is the set of all vectors $\vec{d} = (d_1, \cdots, d_n)$ satisfying $|d_i - d_j| \leq \varepsilon$ and $m \leq d_i \leq M$ $(1 \leq i, j \leq n)$, where $m = \min\{x_1, \cdots, x_n\}$ and $M = \max\{x_1, \cdots, x_n\}$.

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 $\vec{x} = (x_1, \cdots, x_n)$, $T(\vec{x})$ consists of all vectors $\vec{d} = (d_1, \cdots, d_n)$ satisfying: $\vec{d}$ has at most two distinct entries, which lie between the two medics of the multiset $\{x_1, \cdots, x_n\}$, and $d_i \leq d_{i+1} \leq d_i + \varepsilon$.

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. The local processing includes a special operation called *deciding*, which the processor may execute only once; A processor *decides* by writing a *decisive value* to a write-once register.

If the network is initialized with the input vector $\vec{x} \in X^n$ (i.e., the value x_i is assigned to processor P_i), and if each processor executes its own program in a given protocol α, then the sequence of operations performed by the processors is called an *execution of α on input $\vec{x}$*. (We assume here that no two operations occur simultaneously; otherwise, we order them arbitrarily. For more formal definitions see, e.g., [KMZ]. For the definition of the *atomic step* we adapt the model of [FLP].)

Definition: A vector $\vec{d} = (d_1, d_2, \cdots, d_n)$ is an *output vector of α on input $\vec{x}$* if there is an execution of α on $\vec{x}$ in which processor P_i decides on d_i, for $i = 1 \cdots n$.

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 α 1-*solves* a task T if for every execution of α on any 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 $T(\vec{x})$.

When such a protocol α exists we say that the ask 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 have already decided may halt. Hence, in this case, reaching a decision by all non-faulty processors is sufficient to guarantee hating. 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 $t > 1$ crash failures, there exist tasks which can be t-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 asynchrnous fault tolerant protocols. We model an arbitrary t-resilient protocol that work 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 work in rounds of communications, as follows. In each round a processor broadcast 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 (including its own message which is received first; it may wait for less than $n-1$ messages 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 at most one such message per each previous round) are gathered with the $n-1$ messages 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 scud 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 is as follows:

$r \leftarrow 0$
state $\leftarrow$ *INIT_k*
while *state*<> *HALT* **do**
    $r \leftarrow r+1$
    BROADCAST (r, MESSAGE FUNCTION_k (*state*))
    WAIT until you RFEEIVE ($n-1-$[# of known halted processors])
    messsages of the form (r,*)
    $M \leftarrow \{m \mid$ a message (r', m), $r' \leq r$ was received in the above WAIT, or a message (r, m) was received in a previous round$\}$
    state $\leftarrow$ STATE_FUNCTION_k(*state*, M)
end

3.2 Round Complexity

Definition: Let T be a task and α a standard protocol that 1-solves T. The *round complexity of* α *on input* $\vec{x}$ denoted $rc_\alpha(\vec{x})$, is the maximum round

number, over all execution of α on input $\vec{x}$ that a correct processor reaches.

The *round complexity* of α, denoted $rc_\alpha(T)$ is defined by: $rc_\alpha(T) = \sup_{\vec{x}\in X_T} rc_\alpha(\vec{x})$. 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 X_T is infinite, and for any protocol α that 1-solves T and for any constant C, there is an inputy $\vec{x}$ such that $rc_\alpha(\vec{x}) > C$.

Definition: A 1-solvable task T is *bounded* if $rc(T)$ is finite, and is *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 Graphsf Partial Vectors, Covering Vectors and i-Anchors

Definition: Let $S \subseteq A^n$, for a given set A. Two vectors $\vec{s_1}, \vec{s_2} \in S$ are *adjacent* if they differ in exactly one entry. The *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 *decision graph* of $\vec{x}$.

Definition: A *partial vector* is a vector in which one of the entries is not specified; this entry is denoted by '$*$'. For a vector $\vec{s} = (s_1, \cdots, s_n)$, $\vec{s}^i$ denotes the partial vector obtained by assigning $*$ to the i-th entry of $\vec{s}$, i.e., $\vec{s}^i = (s_1, \cdots s_{i-1}, *, s_{i+1}, \cdots, s_n)$. $\vec{s}$ is called an *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 *covering vector* for $\vec{x}^{\,i}$ if for each extension of $\vec{x}^{\,i}$ to an input vector $\vec{x} \in X_T$, there is an extension of $\vec{d}^{\,i}$ to a decision vector $\vec{d} \in T(\vec{x})$.

Note that in an execution on input $\vec{x}$ in which the messages of P_i are delayed, the remaining $n-1$ processors must eventually output a covering vector for $\vec{x}^{\,i}$. If eventually P_i decides too, then the resulted output vector is an *i-anchor*, which we define below.

Definition: A vector $\vec{d}$ is an *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 (defined in Section 2.2) for $n = 3$ processors and $K = 5$. For the partial input vector $\vec{x}^{\,2} = (10, *, 30)$ there is a unique covering vector $\vec{d}^{\,2} = (2, *, 4)$, 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 $\vec{x}^{\,2}$: $(2, *, 4)$, $(2, *, 5)$, and $(3, *, 5)$. Thus, $\vec{x}$ has four 2-anchors: $(2, 3, 4)$, $(2, 3, 5)$, $(2, 4, 5)$ and $(3, 4, 5)$

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*. As we show in Section 6, such functions

may be used to construct protocols that 1-solve T.

Each solving covering function CF defines a *radius function* $\rho_{CF} : X_T \to N$, as follows.

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.

The set of all radius functions for T is denoted by $\mathbf{R}_T$. That is,

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

Definition: $R(T)$, *the radius of the task* T, is given by:

$$R(T) = \min_{\rho_{CF} \in \mathbf{R}_T} \sup_{\vec{x} \in X_T} \rho_{CF}(\vec{x}).$$

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

Note that $R(T)$ may be infinite; This is the case only when the input set X_T is infinite, and for any radius faction ρ_{CF} in $\mathbf{R}_T$ and for any constant C, were is an input $\vec{x}$ such that $\rho_{CF}(\vec{x}) > C$. As we shall show, $R(T)$ is finite iff T is a bounded task.

Example: Consider the following task T for $n = 3$ processors, in which X_T contains 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 action for T, the only pain input vectors that should be considered are those which mist be extent 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 constraint 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.

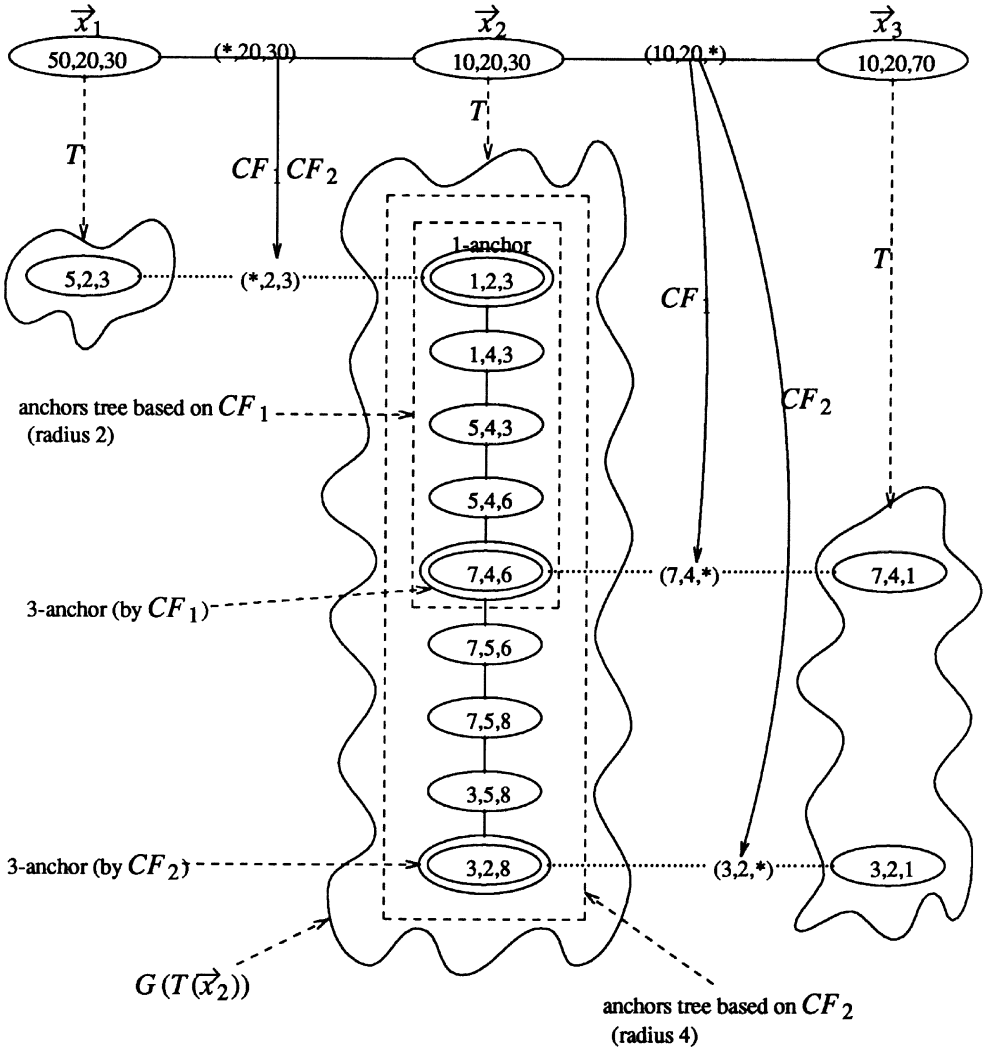

Figure 1: A task T with $R(T) = 2$ $(= \rho_{CF_1}(\vec{x_2}))$

From the decision graphs (see Figure 1), clearly $R(T)$ is determined by $T(\vec{x_2})$, since any anchors tree of the other two input vectors is composed of a single vertex. Based on the previous discussion, it suffices to consider only two covering functions, CF_1 and CF_2, whose values on the two "key" partial

vectors are as follows:
$CF_1(\ (*, 20, 30)\ ) = (*, 2, 3),\ CF_1(\ (10, 20, *)\ ) = (7, 4, *)$ and
$CF_2(\ (*, 20, 30)\ ) = (*, 2, 3),\ CF_2(\ (10, 20, *)\ ) = (3, 2, *)$.
In the minimum radius anchors tree based on CF_1 (in $G(T(\vec{x_2}))$) the 1-anchor is $(1, 2, 3)$, the 3-anchor is $(7, 4, 6)$, and thus the radius is 2 (a line, with center $(5, 4, 3)$). The anchors tree based on CF_2 has the same 1-anchor, its 3-anchor is $(3, 2, 8)$, and its radius is 4. So CF_1 is the optimal covering fbnction, and $R(T) = 2$.

More examples appear in Section 8.

5 LOWER BOUND

In this section we prove the following theorem.

Theorem 1: Let T a bounded task. Then its rood complexity $rc(T)$ satisfies

$$rc(T) \geq \log_{(n-1)} R(T).$$

Proof: Let α be a standard protocol which 1-solves T, and $rc_\alpha(T) = s$. We will prove that α implies a solving covering function for T, CF_α, such that $\rho_{CF_\alpha}(\vec{x}) \leq (n-1)^s$ for every input vector $\vec{x}$ and thus $R(T) \leq (n-1)^s$. To simplify the presentation, we assume that in all executions of α no processor halts before round s (and hence that all processors halt in round s). Clearly, such an assumption does not affect the generality of the proof, since we can always modify α such that processors that halt in round $r < s$ will continue to send "dummy" messages in later rounds. Note that this assumption enables us to assume that in each round, each processor waits for exactly $n - 1$ messages (including its own message) of this round.

For the proof we construct sequences of executions of α, which first require some definitions and discussion.

Definition: e is an *r-rounds execution* of a standard protocol A if e is the first r rounds of an execution of A. e is an *r-rounds i-sleeping execution* if during e, no processor P_j, $j \neq i$, ever receives a message from P_i.

Let e be an r-rounds execution of α, and let $1 \leq l \leq r$. The *l-senders*

of P_k *in* e is the set of processors from which P_k receives messages $(l,^*)$ in the l'th round of e. Note eat the l-senders of P_k always contains P_k, and it its cardinality is $n-1$.

Definition: An r-rounds execution e is an *ordered* execution if for each $1 \leq k \leq n$ and for each $1 \leq l \leq r$, each processor P_k receives in round l exactly all the messages $(t,^*)$, $t \leq l$, sent to him by its l-senders, and which it has not received yet.

All the executions discussed in the rest of this section are ordered executions of the protocol α. Observe that an ordered r-rounds execution e is completely characterized by the inputs to the processors and by specifying the l-senders of each processor, for $l = 1, \cdots, r$.

The *history* of a processor in an r-round execution e of α is defined by its input value, and the messages it receives in each round l from its l-senders, for $l = 1, \cdots, r$.

Proposition 1: Let f and f' be two r-rounds executions of α. Then P_k has the same history in f and f', if (and only if) in both executions it has the same r-senders, S, and each processor of S has the same history after the $(r-1)$-st round in f and f'. (Note that P_k belongs to S.) □

Next we define the solving covering function CF_α. For a given $\vec{x}$ and i, $CF_\alpha(\vec{x}^{\,i})$ is the partial vector $\vec{d}^{\,i}$ output by the processors $P - \{P_i\}$ in an s-rounds i-sleeping execution of α on input $\vec{x}$ (The validity of this definition follows from the fact that α 1-solves T in at most s rounds, and thus by round s the processors $P - \{P_i\}$ must decide on a covering vector).

We now proceeds to the main construction required for our proof, given in Lemma 1 below. This construction uses an idea of [Fe]. First we need the following definition and proposition:

Definition: Two r-roods executions are *adjacent* if there are at least $n-1$ processors, each of which has the same history in both executions.

Proposition 2: Let f and f' be two r-rounds executions which are identical until round $r-1$, and assume there are two processors, each of which has the same r-senders in f and f'. Then there is a sequence of $n-1$ r-rounds

executions, $CHAIN(f, f') = (f = f_1, f_2, \cdots, f_{n-1} = f')$ s.t. f_k and f_{k+1} are adjacent for $k = 1..n-2$.

Proof: For simplicity, assume that the two processors that have the same r-senders in f and f' are P_1 and P_n. Let the r-senders of the processors $P_1, \cdots, P_n$ in execution f be $Q_1, Q_2, \cdots, Q_{n-1}, Q_n$ (Q_i is the r-senders of P_i), and let the r-senders of the processors in execution f' be $Q_1, Q'_2, \cdots, Q'_{n-1}\ Q_n$. Then $CHAIN(f, f') = (f = f_1, \cdots, f_{n-1} = f')$, where for $i = 2, \cdots, n-2$, the first $r-1$ rounds of f_i are identical to those of f and f', and the r senders of the processors $P_1, \cdots, P_n$ in f_i are $Q_1, Q'_2, \cdots, Q'_{i-1}, Q_i, \cdots Q_n$. □

Lemma 1: Let $1 \leq i < j \leq n$ and let $\vec{x} \in X_T$ an input vector. Then for each $r > 0$, there exists a sequence $S_r = e_1, \cdots, e_{D_r}$ of $D_r = (n-1)^r$ r-rounds executions of α, satisfying the following:

(a) e_1 is an r-rounds i-sleeping execution on input $\vec{x}$ and e_{D_r} is an r-rounds j-sleeping execution on input $\vec{x}$.

(b) The executions e_k and e_{k+1} are adjacent, for $k = 1, \cdots, D_r - 1$.

Proof: The proof is by induction on r. (The base and the first step of the induction are depicted in Figure 2). For the base, $r = 1, e_1$ is the 1-round i-sleeping execution on input $\vec{x}$ which the 1-senders of P_i is $P - \{P_j\}$, and e_{n-1} is the 1-round j-sleeping execution on input $\vec{x}$ in which the 1-senders of P_j is $P - \{P_i\}$. The sequence $e_1, \cdots, e_{n-1}$ is $CHAIN(e_1, e_{n-1})$, which satisfies the conditions by Proposition 2 (the assumptions of Proposition 2 hold since P_i and P_j have each the same 1-senders in e_1 and e_{n-1}).

The induction step: Let $S_{r-1} = e_1, \cdots, e_{D_{r-1}}$ be a sequence satisfying the lemma for $r-1$ ($D_{r-1} = (n-1)^{r-1}$). Then for each $k = 1, \cdots, D_{r-1} - 1$ there is a set of $n-1$ processors, which we denote by Q_k, such that each processor in Q_k has the same history in e_k and e_{k+1}.

We construct the sequence S_r by replacing each $(r-1)$-rounds execution e_k in S_{r-1} by a sequence of $n-1$ r-rounds adjacent execution $e_{k,1}, e_{k,2}, \cdots, e_{k,n-1}$. i.e., $S_r = e_{1,1}, \cdots, e_{1,n-1}, \cdots, e_{D_{r-1},n-1}$. It remains to define the executions $e_{k,j}$ and to prove that S_r indeed satisfies the lemma.

First, to avoid special end-case treatments, we define $Q_0 = P - \{P_i\}$ and $Q_{D_{r-1}} = P - \{P_j\}$. Now, in $e_{k,1}$ the first $r-1$ rounds are identical to e_k. In

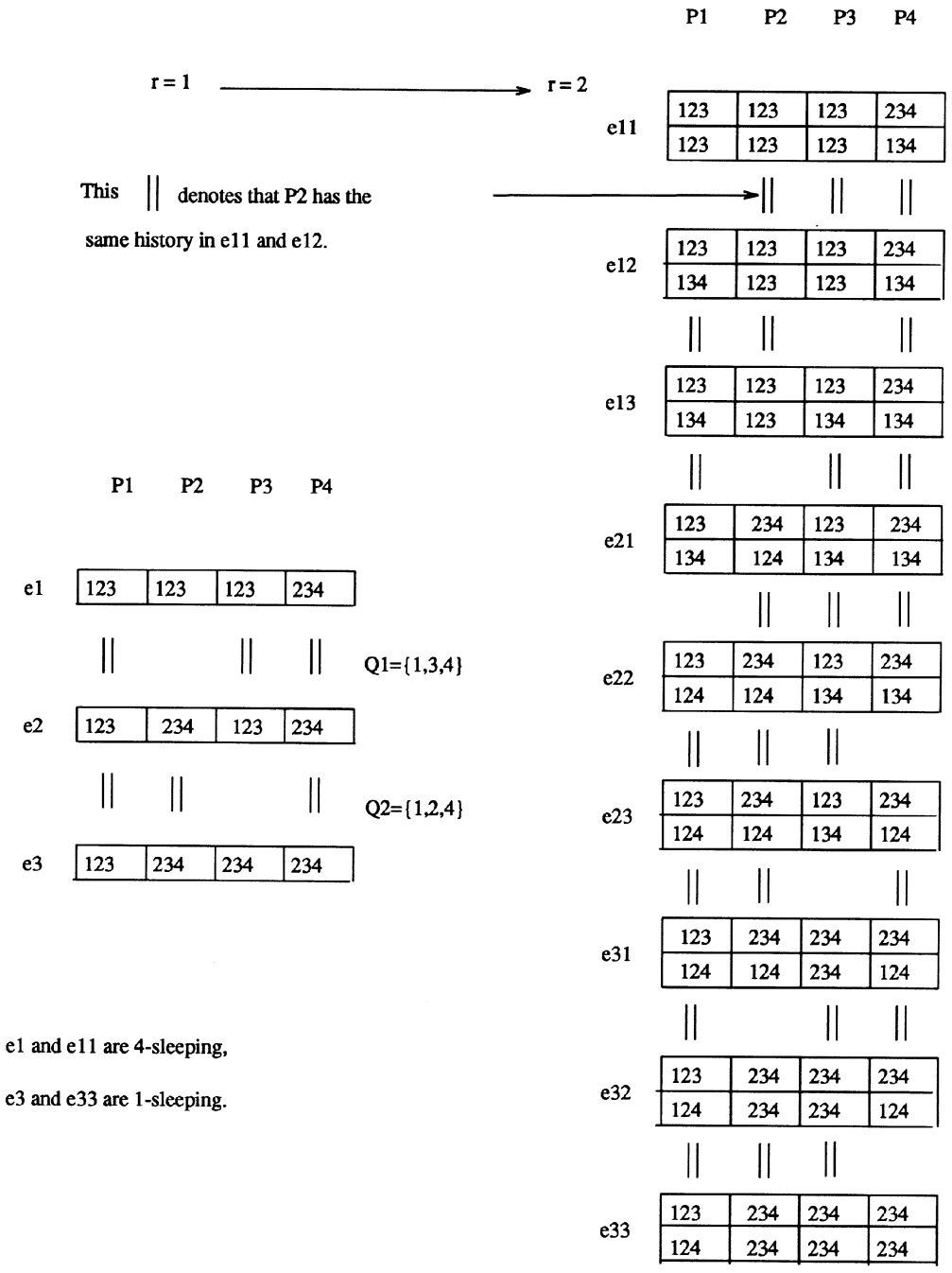

Figure 2: The construction of the sequence for $r = 2$ from the sequence for $r - 1 = 1$; for $i = 4$, $j = 1$

round r: The r-senders of each processor in Q_{k-1} is Q_{k-1}, and the r-senders of the remaining processor is Q_k. In $e_{k,n-1}$ the first $r-1$ rounds are also identical to e_k. In round r: The r-senders of each processor in Q_k is Q_k, and the r-senders of the remaining processor is Q_{k-1}. Now by Proposition 2 we can define the sequence $e_{k,1}, \cdots, e_{k,n-1}$ to be $CHAIN(e_{k,1}, e_{k,n-1})$.

It remains to show that:

1. $e_{1,1}$ (i.e., the lefttnost execution in S_r) is an r-rounds i-sleeping execution and $e_{D_{r-1},n-1}$ (i.e., the impost execution in S_r) is an r-rounds j-sleeping execution. This follows immediately by the induction hypothesis and the construction.

2. Two successive executions in S_r are indeed adjacent. If the two executions are in the same CHAIN (that is, they are $e_{k,i}$ and $e_{k,i+1}$ for some k and i) then this follows from Proposition 2. We now prove that this holds also in the case that these executions are from different CHAINs, i.e. they are of the form $e_{k,n-1}$ and $e_{k+1,1}$ for some k. By the construction, until round $r-1$ $e_{k,n-1}$ is identicaI to e_k and $e_{k+1,1}$ is identical to e_{k+1}. By the induction hypothesis, each processor in Q_k has the same history in e_k and e_{k+1} (and thus has the same history after round $r-1$ in $e_{k,n-1}$ and $e_{k+1,1}$). By the consuction, the r-senders of each processor in Q_k, in both $e_{k,n-1}, e_{k+1,1}$ is Q_k, and thus, by Proposition 1, each of these $n-1$ processors has the same history in $e_{k,n-1}$ and $e_{k+1,1}$. □

We now use Lemma 1 to show that $R(T) \leq D_s = (n-1)^s$. For this, apply Lemma 1 for $r = s$. Then each execution e_k defines an output vector $d_k \in T(\vec{x})$ (since α guarantees that each non-fault processor decides by round s). Statement (a) of the Lemma implies that $\vec{d}_1$ is an i-anchor of $\vec{x}$ which extends $CF_\alpha(\vec{x}^{\,i})$, and $\vec{d}_{D_s}$ is a j-anchor of $\vec{x}$ which extends $CF_\alpha(\vec{x}^{\,j})$. Statement (b) of the lemma implies that for every k, $\vec{d}_k$ and $\vec{d}_{k+1}$ are either the same vector or are adjacent. Thus, $(\vec{d}_1, \cdots, \vec{d}_{D_s})$ is a path of length at most $D_s - 1$ from an i-anchor to a j-anchor of $\vec{x}$. Since this holds for every i and j, $\rho_{CF_\alpha}(\vec{x}) < D_s$. Since $\vec{x}$ is arbitrary, we have that $R(T) \leq D_s$. This completes the proof of Theorem 1. □

6 UPPER BOUND

6.1 The Protocol

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

Proof: We present a protocol that 1-solves T, and whose round complexity is $2 + \lceil \log_{(n-1)} R(T) \rceil$. 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 as possible decision vectors belong to a single path in the anchors tree based on CF. 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 protocol assumes that each processor has a copy of the algorithms CF and $TREE$ above.

The general outline of the algorithm is as follows: In be 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 pain vectors. At his point, there are two any of processors: those who know only partial input vector $\vec{x}^{\,j}$, and hence also know the index j (note that it is the same j for all these processor), 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 $\vec{x}$ start with $R(T)$. During these

rounds, each of the processors that knows the complete input vector $\vec{x}$ and/or the index j, appends to its messages also these values. After these rounds, each processor will have a value v in $[0, R(T)]$ s.t. the difference between the maximal and minimal values is at most 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 $(d_1, \cdots, d_k, \cdots, d_n)$ 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 j. Then, it "normalizes" the value v to an integer q, which is between 0 and the lenght l of the path from the j-anchor to $ROOT(\vec{x})$. Since $l \leq R(T)$, we have that the difference between the maximal and minimal q values is at most 1. Finally, each processor decides on the q-th vector on this path. Since the difference between the q values is 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 $T(\vec{x})$. It is worth mentioning here that deciding on two non-adjacent vectors does not guarantee a legal output vector, and convergence to a single decision vector is actually an agreement, which is impossible by the result of [FLP].

The protocol for P_k:

A. BROADCAST 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. {approximate consensus stage} **if** you how only $\vec{x}^{\,j}$ **then** $v \leftarrow 0$ **else** $v \leftarrow R(T)$
   for $r = 1$ **to** $\lceil \log_{(n-1)} R(T) \rceil$ **do**
   info $\leftarrow \vec{x}$ and / or j (whatever you know of the two)
   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 DECIDE $CF(\vec{x}^{\,j})$
   else if $v = R(T)$ (you know only $\vec{x}^{\,j}$) then DECIDE $ROOT(\vec{x})$
   else (you know $\vec{x}$ and j) **do**
   Let l be the length of the path in $TREE(\vec{x})$ between the j-anchor and $ROOT(\vec{x})$ $q \leftarrow \lfloor vl/R(T) \rfloor$
   DECIDE on the q'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 l)

end

HALT

6.2 Correctness Proof

It is easy to see that each non-faulty processor eventuallly decides. We now assume eat all processors are non-faulty, and prove that the output vector is legal. By the discussion preceding the protocol, it suffices to prove that for each execution of the protocol in which all processors are non-faulty, there are two adjacent vectors such that each processor decides on one of them. If all the processors know the complete input vector $\vec{x}$ at the end of stage B, then all the processors start and finish stage C with $v = R(T)$, and decide at stage D on $ROOT(\vec{x})$, and we are done. Otherwise there exists a unique j such that some processors know only $\vec{x}^{\,j}$ at the end stage B (the uniqueness of j is implied by the fact that $n-1$ is a majority).

Denote by v_k the value of v that P_k holds after $\lceil \log_{(n-1)} R(T) \rceil$ rounds of approximate consensus in stage C, and by q_k the value q it holds after the normalization in stage D. The difference between the maximum and minimum values of the v_k's is at most 1 (since the difference between the v values is initially at most $R(T)$, and it is reduced at least by a factor of $n-1$ each round).

If for all k, $v_k \neq R(T)$ and $v_k \neq 0$, then the each processor P_k computes q_k, and decides on the q_k-th vector on the path from the j-anchor to $ROOT(\vec{x})$. Clearly the maximum difference between the q_k's is 1, since $l \leq R(T)$. Hence, all processors decide on two adjacent vectors on it path.

Otherwise, there are two cases where some processor P_k decides without computing q_k: One case is when $v_k = R(T)$ (and P_k decides on $ROOT(\vec{x})$). In this case all the v_i's are in the range $[R(T)-1, R(T)]$, and the minimum possible q_i is $l-1$, which corresponds to a vector adjacent to $ROOT(\vec{x})$). The other case is when $v_k = 0$, and hence P_k decides on $CF(\vec{x}^{\,j})$. In this case, all the $q_i's$ lie in the interval $[0,1]$, and hence all processors decide on $CF(\vec{x}^{\,j})$, or on the j-anchor, or on a vector adjacent to the j anchor. This case is equivalent to the case where all processors decide on the j-anchor or a vector adjacent to it, since P_j never decides on $CF(\vec{x}^{\,j})$ (it knows its own input x_j), and for every processor other than P_j, deciding on $CF(\vec{x}^{\,j})$ is equivalent to deciding on the j-anchor. □

7 GENERALIZATION

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

Definition: Let T be a 1-solvable task. A function $f : X_T \rightarrow N$ is a *round complexity function of* T if were exists a protocol α that 1-solves T, and for each $\vec{x} \in X_T$, $rc_\alpha(\vec{x}) \leq f(\vec{x})$ ($rc_\alpha(\vec{x})$ 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 order of functions, as follows.

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

We define the set of minimal round complexity function of a task T by a correspondence to the set of minimal radius actions in R_T: we show that for each round complexity function rc there exists a radius action $\rho_{CF} \in \mathsf{R}_T$ s.t. $\log_{(n-1)}\rho_{CF}$ is smaller (or equal) than rc, and for each radius function $\rho_{CF} \in \mathsf{R}_T$, $3 + \lceil \log_{(n-1)}\rho_{CF} \rceil$ is a round complexity faction of T.

Thus, the set of functions

$$\mathsf{mR}_T = \{3 + \lceil \log_{(n-1)}\rho_{CF} \rceil \mid \rho_{CF} \text{ is a minimal function in } \mathsf{R}_T\}$$

approximates the set of minimal round complexity functions of T by an additive constant of 3, in the following meaning: Each function in mR_T is a round complexity function of T s.t. there is no other round complexity function of T which improves it by more than the additive constant 3, and for each minimal round complexity function of T, there is a function in mR_T which is larger by at most 3.

Theorem 1u: Let rc be a round complexity function of a task T. Then, there exists a radius function $\rho_{CF} \in \mathsf{R}_T$ s.t. $\log_{(n-1)}\rho_{CF}(\vec{x}) \leq rc(\vec{x})$, for each

$\vec{x} \in X_T$.

Proof: Since rc is a round complexity function of T, there exists a protocol α s.t. $rc_\alpha(\vec{x}) \leq rc(\vec{x})$ for each $\vec{x} \in X_T$. From this point the proof is so to that of Theorem 1, when s is replaced by $rc_\alpha(\vec{x})$, and the radius function whose existence is proven is ρ_{CF_α}. □

Theorem 2u: Let ρ_{CF} be a radius function of a task T. Then, $3 + \lceil \log_{n-1} \rho_{CF} \rceil$ 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 $\rho_{CF}(\vec{x})$. Now, the problem is that processors that at the beginning of stage C know only $\vec{x}^j$, cannot compute $\lceil \log_{n-1} \rho_{CF} \rceil$ – 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 $n-1v$ values it receives are 0 (and thus still knows only $\vec{x}^j$), 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.

□

8 APPLICATIONS

We present here new optimal bounds on the rood 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 evolved (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 $\log_{n-1} R(T) \leq 1$). Actually, each of the covering function presented will be optimal. The last example deal with the strong binary monotone

approximate consensus, and provide a bound of $4 + \log_{n-1}(\frac{d-c}{\varepsilon})$, where d and c are the two medians of the numbers of the input vector. This is 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. *Strong Binary Monotone Consensus*: Let $\vec{x}^{\,i} = (x_1, \cdots, x_{i-1}, *, x_{i+1}, \cdots, x_n)$ 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 $CF(\vec{x}^{\,i}) = (c, \cdots, c)$, where c is the median of the multiset $\{x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n\}$.

   We now describe anchors trees based on CF. For a given input vector $\vec{x}$ let c and d be the two medians of the multiset $\{x_1, \cdots, x_n\}$. If $c = d$ then the anchors tree consists of the single vertex $(c, \cdots, c)$. Otherwise, it consists of the path $[(c, \cdots, c, d),\ (c, \cdots, c, d, d) \cdots, (c, d, \cdots, d)]$. 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 action 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: $T(\vec{x})$ depends only on the relative order among the entries of $\vec{x}$.

   CF is also order invariant, and we describe $CF(\vec{x}^{\,i})$ only for the case that the entries in $\vec{x}$ are monotone increasing (i.e., $x_i < x_{i+1}$). The adaptation of be definition to other order types is straight forward. In this case, $CF(\vec{x}^{\,i}) = (2, 4, \cdots, 2i - 2, *, 2i, \cdots, 2n - 2)$. A suitable

anchors tree of such $\vec{x}$ is the path of length $2n-2$ (and hence of radius $n-1$) starting at the 1-anchor $(1,2,4,\cdots,2n-2)$ and ending at the n-anchor $(2,4,\cdots,2n-2,2n-1)$, that passes via all the i-anchors. (e.g., for $n=3$ this path is $[(1,2,4),(1,3,4),(2,3,4),(2,3,5),(2,4,5)]$).

4. **Strong Binary Monotone Approximate Consensus** (for a given ε): The input, and the (unique) covering function CF is the same as for the binary monutone consensus. The minimal radius anchors tree based on CF is also similar to the one for the binary consensus, but this time ε must be taken into account:
For a given input vector $\vec{x}$ let c and d be the two medians of the multiset $\{x_1,\cdots,x_n\}$. Assume for simplicity the ε divides $d-c$. If $c=d$ then the anchors tree consists of the single vertex $(c,\cdots,c)$. Otherwise, it consists of the path $[(c,\cdots,c,c+\varepsilon),(c,\cdots,c,c+\varepsilon,c+\varepsilon),\cdots,(c,c+\varepsilon,\cdots,c+\varepsilon),(c+\varepsilon,\cdots,c+\varepsilon),\cdots,(d-\varepsilon,\cdots,d-\varepsilon,d-\varepsilon),\cdots,(d,\cdots,d-\varepsilon)]$. In the first case the radius of the tree is 0, and in the second is $n(\frac{d-c}{\varepsilon})$. Thus, the upper aid provided by our results (for beaded tasks) is at most $5+\log_{n-1}(\frac{d-c}{\varepsilon})$.

References

[ABDKPR] H. Attiya, A. Bar-Noy, D. Dolev, D. Koller, D. Peleg, R. Reischuk, *Achievable cases in an asynchronous environment*, **Proc. of the 28th FOCS**, October 1987, pp. 337–346. A new version in **Journal of the ACM**, Vol. 37 no. 3, 1990, pp. 524–548.

[ADG] H. Attiya, D. Dolev and J. Gil, *Asynchronous Byzantine consensus*, **Proc. of the 3rd PODC**, 1984, pp. 119–133.

[ALS] H. Attiya, N. Lynch and N. Shavit, *Are wait free algorithms fast ?*, **Proc. of the 31th FOCS**, 1990, pp. 422–427.

[BMZ] O. Biran, S. Moran and S. Zaks, *A combinatorial characterization of the distributed task which are solvable in the presence of one faulty processor*, **Journal of algorithms 11**, 1990, pp. 420–440.

[DLS] C. Dwork, N. Lynch and L. Stockmeyer, *Consensus in the presence of partial synchrony*, **Journal of the ACM**, Vol. 35 no. 2, 1988, pp. 288–323.

[DLPSW] D. Dolev, N. A. Lynch, S. Pinter, E. Stark and W. Weihl, *Reaching approximate agreement in the presence of faults*, **Journal of the ACM**, Vol. 33 no. 3, 1986, pp. 499–516.

[Fe] A. D. Fekete, *Asynchronous Approximate Agreement*, **Proc. of the 6th PODC**, 1987, pp. 64–76.

[FL] M. Fisher and N. A. Lynch, *A lower bound for the time to assure interactive consistency*, **Information processing letters** 14:4, 1982, pp. 183–186.

[FLP] M. J. Fischer, N. A. Lynch and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, **Journal of the ACM**, Vol. 32 No. 2 (1985), pp. 373–382.

[HT] M. Herlihy and M. Tuttle, *Wait free computation in message-passing systems*, **Proc. of the 9th PODC**, 1990, pp. 347–362.

[KMZ] E. Korach, S. Moran and S. Zaks, *Tight lower and upper bounds for some distributed algorithm for a complete network of processors*, **Proc. of the 3rd PODC**, 1984, pp. 199–207.

[MW] S. Moran and Y. Wolfstahl, *Extended impossibiliy results for asynchronous complete network*, **Information Processing Letters**, 26, 1987, pp. 145–151.

[NT] G. Neiger and S. Toueg, *Automatically increasing the Fault-tolerance of distributed systems*, **Proc. of the 7th PODC**, 1988, pp. 248–262.

[TKM] G. Taubetield, S. Katz and S. Moran, *Initial failures in distributed computations*, **Journal of parallel programming** 18:4,1989, pp. 255–273.