A COMBINATIONAL CHARACTERIZATION OF THE DISTRIBUTED TASKS WHICH ARE SOLVABLE IN THE PRESENCE OF ONE FAULTY PROCESSOR
(Extended Abstract)

by

O. Biran, S. Moran and S. Zaks

Technical Report #485
December 1987
A COMBINATORIAL CHARACTERIZATION OF THE DISTRIBUTED TASKS WHICH ARE SOLVABLE IN THE PRESENCE OF ONE FAULTY PROCESSOR

(Extended Abstract)

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

ABSTRACT

Fischer, Lynch and Paterson showed in a fundamental paper that achieving a distributed agreement for $N > 1$ processors is impossible in the presence of one faulty processor. This result was later extended by Wolfstahl and Moran who showed that it holds for any task with a connected input graph and a disconnected decision graph (where a vertex in the input [decision] graph is an $N$-tuple of input [decision] values of the processors, and there is an edge connecting two vertices if and only if they differ in exactly one component).

In this paper we extend that latter result, and in fact we set the exact borderline between solvable and unsolvable tasks, by giving a necessary and sufficient condition for a task to be solvable in the presence of a faulty processor. We present a universal protocol which solves any task which is found to be solvable by our condition.

Using our characterization, we derive a novel technique to prove lower bounds on the number of messages that must be sent due to processor failure; specifically, we show that for each fixed $N > 2$ there exist distributed tasks for $N$ processors that can be solved in the presence of a faulty processor, but any protocol that solves them must send arbitrarily many messages in the worst case.
I. INTRODUCTION

An asynchronous distributed network consists of a set of processors, connected by communication lines, through which they may have to communicate in order to accomplish a certain task; the time delay on the communication lines is finite, but unbounded and unpredictable.

In recent years a number of papers that investigate impossibility issues in distributed networks were published. Some of these impossibility results stem from symmetry or from lack of information (like not having distinct identities to the processors or not knowing the size of the network); the works [ASW, FLM] are of this kind. Other impossibility results are due to processors failures that are either naive (e.g., failures of the fail-stop type) or malicious (e.g., Byzantine faults); the works [FLP, MW] and [LSP] are, respectively, of these two types.

In this paper, we study the case when one processor is faulty, which means that all of his messages are not delivered from some point on (fail stop failure). For this case, it was shown in [FLP] that it is impossible to achieve a distributed consensus. This result was extended in several directions. In [DDS] the features of asynchrony that yield the result of [FLP] and related results are analyzed. In [DLPSSW] 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. However, giving a precise characterization of the tasks that can be solved in the presence of $t$ faulty processors remains an interesting open problem. In this paper we provide such a characterization for the case $t = 1$.

The first step towards the result in this paper was done in [MW], where it was shown that any distributed task satisfying a certain combinatorial property is not solvable in the presence of one faulty processor; informally, the input values and the output values of a given problem were described in [MW] by input and output graphs, and it was shown that a distributed task whose input graph is connected and whose output graph is disconnected cannot be solved in the presence of one faulty processor.

In this paper we extend the condition in [MW] to provide a complete characterization of the asynchronous distributed tasks that can be solved in the presence of one faulty processor. This characterization is given in a pure graph-theoretic formulation, in terms of the input and output graphs of these tasks, and the relations between them. A simple protocol that solves tasks satisfying those conditions is also given. Using this characterization, the question whether a given task is solvable in the presence of one faulty processor is reduced, in many cases, to the technical problem of determining certain properties of a given graph. We demonstrate this by extending some known impossibility results to their extremes.

The pure combinatorial properties of our characterization enable us to prove lower bounds on the number of messages needed to solve distributed tasks in the presence of a faulty processor. More specifically, we show that for any fixed $N \geq 3$
there is a distributed task for $N$ processors that can be solved in the presence of one faulty processor, but for every arbitrarily large $M$, there is an input for this task such that any protocol that solves it sends in the worst case more than $M$ messages on this input.

The rest of this extended abstract is organized as follows: In Section 2 we present definitions and notations, and notions like decision tasks, protocols and solvability in spite of one fault (1-solvability) are discussed. In Section 3 we present two conditions (Theorems 1 and 2) that must be met by protocols that 1-solve a given task. These two conditions are then used in Section 4 to derive our main result (Theorem 3), which provides a complete characterization of tasks which are 1-solvable, and we describe a universal protocol that 1-solves such tasks. We conclude in Section 5 with a discussion of the complexity of 1-solvability; we show (Theorem 4) that there exist tasks that can be 1-solved, but such that the executions might be arbitrarily long. In the Appendix we discuss the extensions of the results to anonymous networks. The proofs in this extended abstract are either sketched or omitted; the complete proofs will be given in the full paper.

2. DEFINITIONS AND NOTATIONS

2.1 The Model

We study asynchronous distributed system, called network, that 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 distincts). The outline of the modifications needed in the definitions and the proofs required for this kind of model is given in the appendix. 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 (the exact definition is given in the sequel), in which case messages might not have these properties. The faults discussed in this paper are of the fail-stop type [FLP].

A network of $N$ processors is viewed as an undirected graph with $N$ vertices, each representing a processor. It is implicitly assumed in our proofs that the network is complete, but the results easily generalize to arbitrary biconnected networks, in which a failure of a processor cannot disconnect the network.

2.2 Adjacency Graphs, i-Cliques and Partial Vectors

We first present the basic set-theoretic definitions used in the paper; these definitions are extensions of the ones used in [MW].

Let $A^N$ denote the set of all vectors $\vec{a} = (a_1, a_2, \ldots, a_N)$, where $A$ is an arbitrary set and $a_i \in A$ for every $i$. Let $S \subseteq A^N$. Two vectors $\vec{s}_1, \vec{s}_2 \in S$ are adjacent if they differ in exactly one component. The adjacency graph of $S$,
$G(S) = (S, E_s)$, is an undirected graph, where $(S_1, S_2) \in E_s$ iff $S_1$ and $S_2$ are adjacent.

We denote as a partial vector a vector in which one of the components is not specified; this entry is denoted by '*'. For a vector $S = (s_1, \cdots, s_N)$, $S^i$ denotes the partial vector obtained by assigning * to the $i$-th component of $S$, i.e., $S^i = (s_1, \cdots, \hat{s}_i, *, \cdots, s_N)$. For a set of vectors $S$, $S^i = \{S^i : S \in S\}$.

The following proposition implies a basic relation between partial vectors and cliques in adjacency graphs.

**Proposition:** Let $G(S)$ be the adjacency graph of a set of vectors $S$. Then for each clique $C$ in $G(S)$ there corresponds an integer $i$, $1 \leq i \leq N$, such that all the vectors in $C$ differ from one another in exactly the $i$-th component. Conversely, each set of vectors that differ from one another in exactly the $i$-th component constitute a clique in $G(S)$. □

Let $C$ and $i$ be as defined in the proposition. Then we call $C$ an $i$-clique; each $i$-clique defines a unique partial vector $S^i$ in a natural way. A maximal $i$-clique is an $i$-clique that is not contained in a larger $i$-clique. It follows from the proposition that every partial vector $S^i$ defines a unique maximal $i$-clique, that includes $S$ and all the vectors that differ from $S$ in the $i$-th component only (i.e., that agree with $S^i$); this maximal $i$-clique will be denoted as $C(S^i)$. Note that a maximal $i$-clique that contains more than one vector is also a maximal clique in $G(S)$.

### 2.3 Decision Tasks

In this paper we view a decision task as a mapping of possible inputs to allowable outputs. Let $A$ and $B$ be arbitrary sets. Let$f : A \rightarrow 2^B$ be a function that assigns to each element $a \in A$ a subset $f(a)$ of $B$, and let $C \subseteq A$. We define

$$f[C] = \cup_{c \in C} f(c).$$

Hence, $f[C] \subseteq 2^B$. Note that $f(C) = \{f(c) : c \in C\}$ has a different meaning, and it satisfies $f(C) \subseteq 2^B$.

Let $X$ and $D$ be sets of input values and decision values, respectively. A distributed decision task $T$ is a function

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

where $X_T \subseteq X^N$. $X_T$ is called the input set of the task $T$. The decision set of the task $T$ is the set $D_T = T(X_T)$. Each vector $x = (x_1, x_2, \ldots, x_N) \in X_T$ 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 $d = (d_1, d_2, \ldots, d_N) \in D_T$ 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 $\{(x, d) : x \in X_T \text{ and } d \in T(x)\}$ is recursive.

The adjacency graph $G(X_T)$ of the input set $X_T$ is called the input graph of $T$. The decision graph $G(D_T)$ of $T$ is defined similarly.
Examples:

(1) Consensus [FLP]: A consensus task is any task \(T\) where \(X_T = X^N\) for an arbitrary set \(X\), and such that \(T(x) \subseteq \{(0,0,...,0),(1,1,...,1)\}\) for every input vector \(x \in X_T\). We denote \(\hat{0} = (0,0,...,0)\) and \(\hat{1} = (1,1,...,1)\).

A strong consensus task is a consensus task \(T\), in which there exist two input vectors \(\hat{0}\) and \(\hat{1}\) such that \(T(\hat{0}) = \{\hat{0}\}\) and \(T(\hat{1}) = \{\hat{1}\}\). The consensus task which is implicit in [FLP] is a strong consensus task, with \(X_T =\{0,1\}^N\). A weak consensus task is a consensus task that is not strong.

(2) Approximate Consensus [DLPSW]: This task is defined for any given \(e > 0\). The input set \(X_T = \mathbb{Q}^N\), where \(\mathbb{Q}\) is the set of rational numbers, and for a given input \(x = (x_1,\cdots, x_N)\), let \(m = \min(x_1,\cdots,x_N)\) and let \(M = \max(x_1,\cdots,x_N)\). \(T(x)\) is the set of all vectors \(\hat{d} = (d_1,\cdots,d_N)\) satisfying \(|d_i - d_j| \leq e\) and \(m \leq d_i \leq M\) (1 \(\leq i, j \leq N\).

(3) Order Preserving Renaming (OPR) [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 \((x_1,\cdots,x_N)\) of distinct integers. For a given input \(x\), \(T(x)\) is the set of all integer vectors \((d_1,\cdots,d_N)\) satisfying \(1 \leq d_i \leq K\) and such that for each \(i, j\), if \(x_i < x_j\) then \(d_i < d_j\).

Note: The model in [ABDKPR] assumes that the processors do not have identities. Our results can be modified to hold for this model too, in a way described in the appendix.

2.4 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 reading an input value (an element of \(X\)), sending a message to a neighbor, receiving a message and processing information in the local memory. A processor that completes its program is said to halt. A decision protocol is one in which halting is always associated with writing a decision value (an element of \(D\)). All protocols studied in this paper are decision protocols.

If the network is initialized with the input vector \(x \in X^N\), and if each processor executes its own program, then the sequence of operations performed by the processors is called an execution on input \(x\). (We assume that no two operations occur simultaneously; otherwise, we order them arbitrarily. For more formal definitions see, e.g., [KMZ].) Note that an execution on a given input is not necessarily unique, due to the asynchrony in the network. The set of all the executions of a protocol \(\alpha\) on an input \(x\) is denoted by \(E_\alpha(x)\).

A terminating execution \(e\) of a protocol \(\alpha\) on input \(x\) is an execution of \(\alpha\) on input \(x\) in which all the processors eventually halt. The vector \(\hat{d} = (d_1,d_2,...,d_N)\), where \(d_i\) is the decision value of processor \(p_i\), for every \(i\), is called the output vector of the execution \(e\), and is denoted by \(D_\alpha(e, x)\). \(D_\alpha(x)\) is the set of all output vectors of all the terminating executions \(e\) of the protocol \(\alpha\) on input \(x\). Namely, \(D_\alpha(x) = \bigcup_{e \in TE_\alpha(x)} D_\alpha(e, x)\) where \(TE_\alpha(x)\) is the set of all terminating executions of \(\alpha\) on \(x\). For a set \(S\) of input vectors, \(D_\alpha[S]\) is defined according to (*). Given a protocol \(\alpha\) for a task \(T\), the graph \(G(D_\alpha[X_T])\) is
called the output graph of $\alpha$. Note the difference between this graph, that depends both on a task $T$ and a specific protocol $\alpha$ for it, and the decision graph of the task $T$ (that depends only on $T$).

2.5 Solvability and 1-Solvability

A protocol $\alpha$ solves a task $T$ if for every input vector $x \in X_T$, it satisfies:

1. $E_\alpha(x') = T E_\alpha(x')$ (i.e., all the executions of $\alpha$ on $x'$ are terminating), and
2. $D_\alpha(x') \subseteq T(x')$ (i.e., each execution of $\alpha$ on $x'$ results in a legal output vector).

Note that for every (computable) task $T$ there is a protocol that solves it, by first having each processor send its input to a specified processor, say $P_1$, and then letting $P_1$ decide on some vector $d'$ in $T(x')$ and broadcast it to all other processors.

A processor $P$ is faulty in an execution $e$ if all the messages sent by $P$ during $e$ after a certain time are never received (a fail-stop failure).

Next we define the notion of solvability in spite of one fault. We adapt the approach in [MW].

A protocol $\alpha$ 1-solves a task $T$ if the following two conditions hold:

1. If no processor is faulty then $\alpha$ solves $T$, and
2. If, in an execution $e$, one processor is faulty, then all other processors eventually halt.

In this case we say that the decision task $T$ is 1-solvable.

The strong consensus tasks are shown in [FLP] not to be 1-solvable. The weak consensus tasks are clearly 1-solvable, by simply letting every processor decide on 1 (or 0), regardless of the input. The reason is that our definition of solvability does not exclude such trivial solutions. Note that our definition differs from the ones in [FLP,MW] in that we do not require from a protocol that solves $T$ to achieve every possible output. We use this definition since we believe it is more natural: usually, one considers a task solved by any protocol that always produces an acceptable output vector, regardless of whether there are some allowable decision vectors that are never achieved.

3. TWO BASIC CONDITIONS FOR 1-SOLVABILITY

Given a task $T$ and a protocol $\alpha$, we present in this section two necessary conditions for the 1-solvability of $T$ by $\alpha$. We first discuss the connectivity condition, that implies that if $\alpha$ 1-solves $T$ then for each input $x'$, the graph $G(D_\alpha(x'))$ must be connected. Next we discuss the sleeping processor condition, that implies that if $T$ is 1-solvable then $G(D_T)$ must contain cliques of certain type. These conditions are later used to obtain a complete characterization of 1-solvable tasks.
3.1 The Connectivity Condition

Our first theorem is a straightforward generalization of Theorem 3 in [MW], which we state here using our notation:

Theorem MW: Let \( T \) be a decision task that have a connected input graph, and let \( \alpha \) be a given protocol. If \( \alpha \) 1-solves \( T \), then \( G(D_{\alpha}[X_T]) \) is connected.

Our first theorem extends Theorem MW to cases where only a sub-task of the given task satisfies the assumption of the theorem.

Theorem 1 (the Connectivity Condition): Let \( T \) be a decision task, let \( C \subseteq X_T \) be such that \( G(C) \) is a connected subgraph of the input graph \( G(X_T) \), and let \( \alpha \) be a given protocol. If \( \alpha \) 1-solves \( T \), then \( G(D_{\alpha}[X_T]) \) is connected.

Proof. Let \( \alpha \) be a protocol that 1-solves the task \( T: X_T \rightarrow D_T \). Define a new task \( T': C \rightarrow D_{\alpha}[C] \), such that \( T'(x') = T(x') \) for every \( x' \in C \). Clearly 1-solves \( T' \). Moreover, by the definition of the mapping \( D_{\alpha} \) we have \( D_{\alpha}[C] = D_{\alpha}[X_T] \), and by applying Theorem MW to \( T' \) we have that \( G(D_{\alpha}[X_T]) \) is connected. \( \square \)

We shall use in the sequel the following corollary of Theorem 1, for which we need the following definitions:

Definition: A task \( T' \) is a restriction of a task \( T \) if \( X_{T'} = X_T \), and \( T'(x) \subseteq T(x) \) for every \( x \in X_T \).

Note that if \( \alpha \) is a protocol which (1-)solves \( T' \), then \( \alpha \) also (1-)solves \( T \).

Definition: Let \( T \) be a task and \( \alpha \) a protocol which solves \( T \). We denote by \( T^\alpha \) the task induced by \( \alpha \) and \( X_T \); that is: \( X_{T^\alpha} = X_T \), and \( T^\alpha(x) = D_{\alpha}(x) \) for every \( x \in X_{T^\alpha} \).

Note that \( T^\alpha \) is a restricted task of \( T \).

Definition: A task \( T \) is pointwise connected if for each \( x \in X_T \) it holds that \( G(T(x)) \) is connected.

Corollary 1: If a protocol \( \alpha \) 1-solves a task \( T \) then \( T^\alpha \), the task induced by \( \alpha \) and \( X_T \), is pointwise connected.

3.2 The Sleeping Processor Condition

The following theorem is based on the observation that, by the definition of 1-solvability, if we delay all the messages sent by any processor \( P_i \) for long enough, then all other processors must eventually halt. Moreover, the decisions they make (knowing only \( N-1 \) input values) must be extendible to an acceptable decision vector.

Definition: Let \( C(x^i) \) be a maximal input i-clique in \( G(X_T) \) and \( B \) an i-clique in \( G(D_T) \). We say that \( B \) is a covering clique for \( C(x^i) \) (with respect to the task \( T \)) if \( T(y^i) \cap B \neq \emptyset \) for every \( y^i \in C(x^i) \). The partial decision vector defined by a covering clique for \( C(x^i) \) is called a covering partial vector for \( C(x^i) \).

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 halt and decide (such an execution exists by the definition of 1-solvability, since \( P_i \) is not distinguishable from a faulty processor).
Theorem 2 (the Sleeping Processor Condition): Let \( T \) be a decision task and \( \alpha \) be a protocol that \( 1 \)-solves \( T \). Then in \( T^\alpha \), the task induced by \( \alpha \) and \( X_T \), there is a covering clique for each maximal input i-clique.

Proof: Let \( C(T^i) \), where \( T^i = (x_1, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots, x_N) \), be a maximal input i-clique. Consider an i-sleeping execution of \( \alpha \), in which the input to \( P_j \) is \( x_j \) for each \( j \neq i \). Let \( d^i = (d_1, \cdots, d_{i-1}, d_i, d_{i+1}, \cdots, d_N) \) be the partial vector output by the non-sleeping processors. We claim that the maximal i-clique \( D = C(d^i) \) in \( G(D_T^\alpha) \) is a covering clique for \( C(T^i) \).

Let \( y_i \) be any value such that the vector \( y = (x_{i-1}, y_i, x_{i+1}, \cdots, x_N) \) is a possible input vector in \( C(y_i) \). We must show that \( D \cap T^\alpha (y) \neq \emptyset \). This is not difficult to see that \( D \cap T^\alpha (y) = \emptyset \).

Examples: Consider the OPR task for 3 processors and \( K=4 \). Consider the maximal input 3-clique \( C(V^3) \) where \( V^3 = (10,12,*) \). Consider the following input vectors in \( C(V^3) \): \( x_1 = (10,12,13), x_2 = (10,12,11), x_3 = (10,12,9) \). By the definition of \( T \), \( T(x_1) = \{(1,2,3),(1,2,4),(1,3,4),(2,3,4)\} \), \( T(x_2) = \{(1,3,2),(1,4,2),(1,4,3),(2,4,3)\} \) and \( T(x_3) = \{(2,3,1),(2,4,1),(3,4,1),(3,4,2)\} \). Hence \( T(x_1)^3 = \{(1,2,*),(1,3,*),(2,3,*)\} \), \( T(x_2)^3 = \{(1,3,*),(1,4,),(2,4,\*)\} \), \( T(x_3)^3 = \{(2,3,\*),(2,4,\*),(3,4,\*)\} \) and \( (T(x_1)^3 \cap T(x_2)^3 \cap (T(x_3)^3 = \emptyset \). This implies that \( T_E(C(V^3)) = \emptyset \), so by Theorem 2 this task is not \( 1 \)-solvable.

Now examine the same task, but with \( K=5 \). For the same three input vectors we have \( T(x_1)^3 = \{(1,2,\*),(1,3,\*),(2,3,\*)\} \), \( T(x_2)^3 = \{(1,3,\*),(1,4,\*),(2,4,\*),(1,5,\*),(2,5,\*),(3,5,\*)\} \) and \( T(x_3)^3 = \{(2,3,\*),(2,4,\*),(3,4,\*),(2,5,\*),(3,5,\*),(4,5,\*)\} \). Then \( (T(x_1)^3 \cap (T(x_2)^3 \cap (T(x_3)^3 = \{(2,4,\*)\} \), and indeed in this task, if messages sent by some processor are delayed, the other two processors may eventually decide on 2 and 4, so that for each possible input of the delayed processor there is a "room" for its decision (1,3 or 5), and in fact \( T_E(C((10,12,\*)) = \{(2,4,\*)\} \). In this case there is only one covering clique, namely \( \{(2,4,1),(2,4,3),(2,4,5)\} \), which is the maximal clique defined by the above partial vector. One can easily extend this example to show that the minimum \( K \) required for the OPR task is \( 2N - 1 \) (\( N-1 \) decisions and \( N \) "rooms" between them). This result is presented in [ABDKPRJ], where it is also showed that this condition suffices for the task to be \( 1 \)-solvable.

4. A NECESSARY AND SUFFICIENT CONDITION FOR 1-SOLVABILITY

In this section we combine the necessary conditions given in Theorems 1 and 2 to give a a necessary and sufficient condition for a task to be \( 1 \)-solvable. In proving the positive direction we present a universal protocol which \( 1 \)-solves any such task.
Theorem 3: A task $T$ is 1-solvable if and only if there exists a restriction of $T$, $T'$, satisfying the following:

1. $T'$ is pointwise connected, and

2. For each maximal input $i$-clique $C(x^i)$ there is a covering clique in $T'$; moreover, there is a (centralized) algorithm that on input $x^i$ outputs a $d^i$ so that $C(d^i)$ is such a covering clique.

Proof:

Only If: Let $\alpha$ be a protocol which 1-solves $T$. We claim that $T^\alpha$, the task induced by $\alpha$ and $X_T$, has the desired properties: $T^\alpha$ is pointwise connected by Corollary 1, and it contains a covering clique for each maximal input $i$-clique by Theorem 2; moreover, for each partial input vector $x^i$, the corresponding $d^i$ can be computed by simulating an $i$-sleeping execution of $\alpha$ on input $x^i$, as described in the proof of Theorem 2.

Before proving the if part, we give an example to illustrate the proof above: Consider the OPR task with $N=3$, $K=5$. By [ABDKPR] this task is 1-solvable. We show below that if we add to it the requirement that each decision vector must have 2 as one component and 4 as another component, then the resulting task is not 1-solvable.

Consider the input vector $x = (10,20,30)$. By the definition of this task, $T(x) = \{(1,2,4),(2,3,4),(2,4,5)\}$. Also, $x$ participates in 3 maximal input $i$-cliques, defined by the partial vectors $x^1 = (\ast,20,\ast)$, $x^2 = (10,\ast,\ast)$ and $x^3 = (10,20,\ast)$. To each of these $i$-clique there is a unique covering clique in $G(D_r)$, given by ($D_i$ denotes the covering clique for $C(x^i)$):

$D_1 = \{(1,2,4),(2,3,4),(2,4,5)\}$, $D_2 = \{(1,2,4),(2,3,4)\}$ and $D_3 = \{(2,4,1),(2,4,3),(2,4,5)\}$.

Therefore, for any restriction $T'$ of $T$ that satisfies condition (3b), it must hold that $D_i \cap T'(x) \neq \emptyset$ for $i = 1,2,3$. This implies that $T'(x^i) = T(x^i)$. But $G(T(x))$ is not connected, and hence $T'$ is not pointwise connected. It follows that this task is not 1-solvable.

We now complete the proof of Theorem 3:

If: Let $T$ be a task which has a restriction $T'$ satisfying (3a) and (3b). We will present a protocol which 1-solves $T'$, and hence 1-solves $T$.

Definition: Let $x^i$ be an input vector, and let $D = C(x^i)$ be a covering clique for $C(x^i)$. Then an output vector $d^i$ is an $i$-anchor of $x^i$ if $d^i \in T'(x^i) \cap D$. Informally, $i$-anchors are those decision vectors that are output in $i$-sleeping executions.

By condition (3b), there is an algorithm COMP.CLIQ that gets as input a partial input vector $x^i$ and outputs a partial decision vector $d^i$ such that $C(d^i)$ is a covering clique for $C(x^i)$. By (3a) $G(T'(x^i))$ is connected. Hence, for a given finite set $S_x$ of $i$-anchors of $x^i$, there is a finite tree $T_{S_x}$ in $G(T'(x^i))$ that contains $S_x$. It is not hard to show, by the computability of $T$, that there is an algorithm COMP.TREE that on input $x^i$ and a (finite) set $S_x$ of $i$-anchors of $x^i$ outputs a tree $T_{S_x}$ as above and a root $r_x$, which is an (arbitrary) vertex in $T_{S_x}$.
Our protocol assumes that each processor \( P_k \) has a copy of the algorithms \( \text{COMP.CLIQ} \) and \( \text{COMP.TREE} \) described above.

The general outline of the algorithm is as follows: At the first two stages each processor \( P_k \) is trying to find out the input vector \( \mathbf{x}_k \); for this, it first broadcasts its input value and receives \( N-1 \) input values (including its own), which determine a partial input vector \( \mathbf{x}_k^i \) (note that \( i \neq k \)). Then it broadcasts this partial vector, and waits until it receives \( N-1 \) such partial vectors; if all these \( N-1 \) partial vectors are equal to its own, \( \mathbf{x}_k^i \), then \( P_k \) decides on the output vector \( \mathbf{d}_k^i \equiv \text{COMP.CLIQ}(\mathbf{x}_k^i) \) (by saying that \( P_k \) decides on a (partial) output vector \((d_1, \ldots, d_k, \ldots, d_N)\) we mean that it decides on output value \( d_k \)). It then broadcasts \( \mathbf{d}_k^i \) to all other processors, and halt.

Otherwise, \( P_k \) knows the input vector, and it computes the tree \( TR_x = \text{COMP.TREE}(\mathbf{x}, \mathbf{A}_1^i, \ldots, \mathbf{A}_N^i) \), where \( \mathbf{A}_i^i \) is an \( i \)-anchor of \( \mathbf{x} \) in \( C(\text{COMP.CLIQ}(\mathbf{x}_i^i)) \).

In the rest of the algorithm each processor attempts to decide on a certain vertex in \( TR_x \), such that eventually every processor will decide on one out of two adjacent vertices. This is done in phases, where at phase \( l \) each processor suggests a certain vertex \( \mathbf{d}^l \) as a possible decision, by broadcasting \( \mathbf{d}^l \) to all other processors. If at phase \( l+1 \) it receives \( N-1 \) messages of phase \( l \) suggesting the same vertex \( \mathbf{d}^l \), then it decides on \( \mathbf{d}^l \), broadcast its decision, and halts. If it receives \( N-2 \) messages suggesting some vertex \( \mathbf{d}' \) then, suspecting that some other processor might have received \( N-1 \) messages suggesting \( \mathbf{d}' \), it suggests at phase \( l+1 \) the father of \( \mathbf{d}' \) in \( TR_x \) as a possible decision. If it receives less then \( N-2 \) identical messages, then it suggests at phase \( l+1 \) the father of the vertex it had suggested at phase \( l \). This process guarantees that if for long enough time no vertex was decided-upon, all the processors will eventually suggest the root \( r_x \) (which is defined to be its own father), and then will decide on it. A formal description of the protocol is given below; in this protocol we use the procedures \( \text{COMP.CLIQ} \) and \( \text{COMP.TREE} \) described above. It is assumed in this protocol that \( N > 3 \). The protocol for \( N = 3 \) is a little more involved, and will be described in the full paper.

The protocol for \( P_k \):

A. broadcast your input value \( x_k \) and wait until you receive \( N-1 \) stage-A messages.

B. now for some \( j \) (\( 1 \leq j \leq N \)) you know \( x_j^i \) (note that \( j \neq k \).) broadcast \( x_j^i \), and wait until you receive \( N-1 \) stage-B messages.  

\[ l \leftarrow 1 \quad \text{[l is the phase number]} \]

begin  

\[ \mathbf{d}^l \leftarrow \text{COMP.CLIQ}(\mathbf{x}_i^i) \quad \text{[} \mathbf{d}^l \text{ is a partial output vector s.t. } C(\mathbf{d}^l) \text{ is a covering clique for } C(\mathbf{x}_i^i) \text{]} \]

\[ \text{DECIDE}(\mathbf{d}^l) \quad \text{[decide on the } k \text{-th component of } \mathbf{d}^l \]  

\[ \text{BROADCAST(DECIDE, } \mathbf{d}^l, \star) \]  

HALT;  

end;

else \{now you know the input vector \( x \)\}  

\[ TR_x \leftarrow \text{COMP.TREE}(\mathbf{x}, \mathbf{A}_1^i, \ldots, \mathbf{A}_N^i) \quad \text{[} \mathbf{A}_i^i \text{ is an } i \text{-anchor of } \mathbf{x} \text{ in } C(\text{COMP.CLIQ}(\mathbf{x}_i^i)) \text{]} \quad \text{[} r_x \text{ denotes the root of } TR_x \]  

if you received \( N-2 \) messages \( (\mathbf{x}_i^i) \) then \{suspect that some other processors received \( N-1 \) such messages\}  

\[ d^l \leftarrow \text{FATHER}(\mathbf{A}_i^i) \quad \text{[} d \text{ gets the father, in } TR_x \text{, of the } s \text{-anchor of } \mathbf{x}_i^i \]
C. while NOT decided do
   begin
      \( I \leftarrow I + 1; \)
      RECEIVE \( N-1 \) messages of phase \( I-1; \) ([DECEDE, \( d', * \)] messages are considered to be of phase \( m \) for every \( m \))
      if one of the messages is a (DECEDE, \( d', * \)) or all \( N-1 \) messages are (SUGGEST, \( d', I-1 \)) then
         begin
            DECIDE (\( d' \));
            BROADCAST (DECEDE, \( d', * \));
            decided \( \leftarrow \) true;
         end
      else if \( N-2 \) messages are (SUGGEST, \( d', I-1 \))
         then \( d' \leftarrow \) FATHER (\( d' \));
         else \( d' \leftarrow \) FATHER (\( d \));
      BROADCAST (SUGGEST, \( d', I \));
   end.

The correctness of the protocol above follows from the following claims. For a given input vector \( x \), \( TR_x \) and \( r_x \) are the tree containing the \( i \)-anchors of \( x \) and its root, as defined above. \( L_x \) denotes the maximal distance, in this tree, from an \( i \)-anchor to the root \( r_x \).

Claim 1: In each execution of the protocol there is an \( I \) such that at least one processor DECIDEs at phase \( I \). \( \Box \)

Let \( I_0 \) be the minimal \( I \) satisfying Claim 1, let \( P_k \) be a processor that decides in phase \( I_0 \), and let \( d' \) be the vertex it decides on (the case where it decides on a partial vector \( d' \) (at stage B) is similar).

Claim 2: If some processor \( P_j \) decides in phase \( I_0 \) on a vertex \( d' \), then \( d' = d \). \( \Box \)

Claim 3: Exactly one of the following occurs:

(a) At least two processors send at phase \( I_0 \) a (DECIDE, \( d', * \)) message, or

(b) All the processors, except \( P_k \), send at phase \( I_0 \) a message (SUGGEST, FATHER(\( d' \), \( I_0 \))). \( \Box \)

Finally, from Claim 3 and step C of the protocol we get:

Claim 4: For \( j = 1, \cdots, N \), processor \( P_j \) decides at phase \( I_0 \) or at phase \( I_0 + 1 \) on \( d' \) or on FATHER(\( d \)). \( \Box \)

The proof of the correctness of the protocol is easily derived from the above claims and the observation that if all processors decide on two adjacent vertices then the vector they output is one of these vertices. This completes the proof of Theorem 3. \( \Box \)

We demonstrate the use of Theorem 3 by two examples, in which we extend two known 1-solvability results to their extremes. For this, all we have to do is to show existence of a restriction \( T' \) of \( T \) satisfying Theorem 3:

Strong Approximate Consensus: This task is defined similarly to the Approximate Consensus, with the following stronger restriction: For a given input \( x = (x_1, \cdots, x_N) \), let \( a \) be the average of the \( N-1 \) smallest \( x_i \)'s, let \( A \) be the average of the \( N-1 \)
largest $x_i$'s. $T(x)$ is restricted to the set of all vectors $d=(d_1, \ldots, d_N)$ satisfying $|d_i - d_j| \leq \varepsilon$ and $a \leq d_i \leq A$ ($1 \leq i, j \leq N$). Note that this is the most severe restriction that can be imposed on the range of the output numbers. In fact, for a partial input vector $x'=(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N)$ there is only one covering partial vector $d'$, where $d'$ is the "constant" partial vector $(d, \ldots, d, \ldots, d)$, $d$ being the average of the $N-1$ inputs $\{x_j: j \neq i\}$. It is not hard to verify that this task satisfies the conditions of Theorem 3 (with $T' = T$), and hence is 1-solvable.

We can further restrict this task, by requiring that the output vectors are monotone, that is, $d_i \leq d_{i+1}$. The resulting decision graph still satisfies Theorem 3, and hence the resulting task, called the monotone strong approximate consensus, is still 1-solvable.

We shall show later that the number of messages needed to solve the above versions of approximate consensus is not bounded by any fixed constant.

Restricted OPR with $N=3$ and $K=5$: This version is similar to the OPR, but it is required that the difference between the maximal and minimal components of the output vector never exceeds 3. In fact, we can restrict the decision set for each input vector to include only 5 decision vectors. Fig. 1a shows the corresponding graph $G(T(x))$ for $x=(A, B, C)$ with $A < B < C$. The three marked vectors are the anchors of such input. For comparison, Fig. 1b shows the corresponding graph for the original formulation of the problem. In fact, choosing any connected subgraph of this latter subgraph, that includes all the $i$-anchors, defines a version of the OPR which is 1-solvable. This example can be generalized to the OPR with $N$ processors and $K=2N-1$, to define a restricted version which allows only $2N-1$ possible output vectors for each input vector (see Fig. 1c for the case $N=4$), compared to the $(2N-1)^N$ which are allowed by the original formulation.

5. LOWER BOUNDS

Once we have characterized the 1-solvable tasks, it is natural to ask about the cost of their solutions. A natural measure for this cost is the message complexity of such a task, that is: the number of messages that must be sent in the worst case by any protocol that 1-solves it. Note that if all the processors are non faulty, then every computable task can be solved by $O(N)$ messages. In this section we use the characterization theorem given in the previous section to prove the following:

Theorem 4: For a given $N \geq 3$, there is a 1-solvable distributed task $T$ for $N$ processors satisfying the following: For every arbitrarily large constant $M$, there is an input $x$ to $T$, such that every protocol that 1-solves $T$ must send, in the worst case, at least $M$ messages on input $x$.

The proof of the above theorem is based on first showing that every protocol $\alpha$ that sends at most $M$ messages on input $x$ must satisfy $|D_{\alpha}(x)| < F(M)$ for some function $F$, and then using Theorem 3 to show that if $\alpha$ 1-solves $T$ then $|D_{\alpha}(x)| > F(M)$.
To simplify the discussion we consider in this section only executions that satisfy the FIFO discipline on each communication link. Clearly, if a protocol 1-solves a task \( T \), then it must solve it also under this restricting assumption. The converse is also true, i.e., if a task \( T \) is 1-solvable by protocols that assume the FIFO discipline, then it is also 1-solvable by protocols that do not assume it, since this discipline can be simulated by having each processor \( p_i \) number the messages it sends to each processor \( p_j \) (note that for this simulation to work we need the assumption that once a message sent by \( p_i \) is lost, all the following messages sent by it are lost too). Hence, it is not hard to see that any lower bounds that assumes this discipline is also applicable in the case where this discipline is not assumed.

**Lemma 1:** Let \( \alpha \) be a protocol that 1-solves a task \( T \), and let \( x \) be in \( X_T \). Then if at most \( M \) messages are sent in any (FIFO) execution of \( \alpha \) on \( x \), then \( |D_{\alpha}(x)| < (N+1)^2M \). □

**Lemma 2:** There exist tasks \( T \) such that for each arbitrarily large \( M \) there exist an input vector \( x \), such that the distance between any 1-anchor and any 2-anchor of \( x \) is greater than \((N+1)^2M\). □

**Proof of Theorem 4:** Let \( T \) be a task satisfying Lemma 2, and let \( M \) be given. Let \( x \) be an input vector whose existence is guaranteed by Lemma 2. Then by the proof of the only if part of Theorem 3 we know that every protocol \( \alpha \) that 1-solves \( T \) must satisfy that \( G(D_{\alpha}(x)) \) is connected and it contains an \( i \)-anchor of \( x \) for \( i = 1, 2, \ldots, N \). By Lemma 2, this implies that \( |D_{\alpha}(x)| > (N+1)^2M \). By Lemma 1, this implies that \( \alpha \) may send more than \( M \) messages on input \( x \). □

**Example:** One task for which the above lower bound is applicable is the Strong Approximate Consensus, for any fixed \( \epsilon \). To see this, let \( \epsilon = 1 \), and consider an input \( x = (B(N-1), -B(N-1), 0, \ldots, 0) \), where \( B \) is some sufficiently large constant. Then every 1-anchor of \( x \) is of the form \( A^i_1 = (A, -B, -B, \ldots, -B) \), where \( |A + B| < \epsilon \), and every 2-anchor of \( x \) is of the form \( A^2_2 = (A, B, B, \ldots, B) \), where \( |A - B| < \epsilon \), and the distance in \( G(D_T) \) between any two such anchors is approximately the distance between them in the \( l_1 \) metric, namely \( 2NB \). Since \( B \) can be taken to be arbitrarily large, this task satisfies Lemma 2, and hence Theorem 4.

**REFERENCES**


Figure 1a: $G(T(x'))$ in the restricted OPR ($N=3, K=5$) for $x'=(A, B, C)$ s.t. $A < B < C$. The marked vertices are the $i$-anchors.

Figure 1b: The original OPR decision graph for the same input vector. Any connected subgraph that includes the three $i$-anchors defines a 1-solvable restricted version of the OPR task.

Figure 1c: The similar restricted OPR for $N=4$ ($K=7$).
APPENDIX

We sketch below the modifications needed in our definitions and proof in order for our results to hold for the model in which the processors have no identities and the inputs are distinct (note that this includes the case where the processors have distinct identities which are not mutually known, and the input is arbitrary).

In this case $N$ distinct input values are viewed as a subset of $N$ elements of the input set $X$, and the input set of a task $T, X_T$, is a collection of such subsets of $X$.

Let $D$ be the decision set. We denote by $D^N$ the collections of all sets $\{(x_1, d_1), \cdots, (x_N, d_N)\}$, where the $x_i$'s are distinct elements of $X$ and the $d_i$'s are in $D$, $i = 1, \cdots, N$.

A task $T$ in this model is a function $T : X_T \rightarrow 2^{D^N} \setminus \{\emptyset\}$. Informally, for each $x = (x_1, x_2, \cdots, x_N) \in X_T$, $\{(x_1, d_1), (x_2, d_2), \cdots, (x_N, d_N)\}$ is in $T(x)$ iff the assignment of the decision value $d_i$ to the input value $x_i$ ($i = 1, \cdots, N$) is valid for $T$, when the input set is $x$.

The decision set of $T$, $D_T = T[X_T]$ is defined as in section 2.3.

The vertices of the input and decision adjacency graphs, $G(X_T)$ and $G(D_T)$, are now sets of cardinality $N$, as described above. There is an edge connecting two such vertices iff they differ in exactly one element.

The definition of a covering clique for maximal input clique is similar to the one in section 3.2, where "vectors" are replaced by "sets". In particular, a maximal input clique is denoted by $C(x^*)$ where $x^*$ is a subset of $N-1$ elements of $x$.

The equivalent of Theorem 3 in this case is:

Theorem 3': A task $T$ is 1-solvable if and only if there exists a restriction $T'$ of $T$ satisfying the following:

(3'a) $T'$ is pointwise connected, and

(3'b) For each maximal input clique $C(x^*)$ there is a covering clique in $T'$; moreover, there is a (centralized) algorithm that on input $x^*$ outputs a $d^*$ so that $C(d^*)$ is such a covering clique.

In the proof of Theorem 3', we first show that a task $T$ satisfies conditions (3'a) and (3'b), iff by assigning distinct identities to the processors and using them to represent the task in the "vectorial" notation, we get a task that satisfies condition (3a) and (3b). This immediately implies the only if part.

In proving the if part, all that is needed is to adjust the operations in the universal protocol at section 4 to the new definitions in a straightforward manner. It should be pointed out that, since the inputs are distinct, we may assume, as before, that all the processors that received all the input values will agree on the same anchors, and will construct the same tree $T_{R_x}$ used in the protocol.