DECIDING I-SOLVABILITY OF DISTRIBUTED TASKS IS NP-HARD.

by

Ofer Biran, Shlomo Moran and Shmuel Zaks

Technical Report #666
March 1991
DECIDING 1-SOLVABILITY OF DISTRIBUTED TASKS IS NP-HARD¹

Ofer Biran, Shlomo Moran and Shmuel Zaks

Department of Computer Science
Technion, Haifa, Israel 32000

ABSTRACT

The question of which distributed tasks can be solved by asynchronous protocols in the presence of crash failures has been widely investigated in recent years. Most of these studies investigate the solvability of specific tasks, providing both positive and negative results. However, the general question whether there is an algorithm that receives as an input a specification of a distributed task \( T \) for \( N \) processors and a number \( r \), and decides whether \( T \) is solvable in the presence of at most \( r \) crash failures, was less studied. In fact, we are not aware of any published proof that this problem is decidable, even when restricted to finite tasks (which are given by a finite list of input-output relations).

In this paper we provide partial results concerning this question. For this, we use a result in [BMZ], which associates with each distributed task certain graphs, that enable us to determine whether this task can be solved in the presence of one faulty processor. We use that characterization to show that the above problem is NP complete for \( r=1 \) and \( N \geq 3 \), and NP hard for any fixed \( r \) and \( N > 2r \).

¹ A preliminary version of this paper appeared in the proceedings of the 16th International Workshop on Graph-Theoretic Concepts in Computer Science, Berlin, June 1990. This research was supported in part by Technion V.P.R. Funds - Weizman Research Fund and Loewengart Research Fund, by the I Goldberg Fund for Electronics Research, and by the Foundation for Research in Electronics, Computers and Communications, administered by the Israel Academy of Sciences and Humanities.
1. INTRODUCTION

The question of which distributed tasks can be solved by asynchronous protocols in the presence of crash failures has been widely investigated in recent years. In [FLP] it was shown that the consensus task cannot be solved in the presence of one fail-stop failure in the asynchronous model. In [DLPSW] it was shown that the approximate consensus is solvable in the presence of multiple such failures, and in [ABDKFR] two versions of the renaming task were studied. Other cases were studied in [BW], in which a class of tasks were proven not to be solvable when one processor is faulty. A complete characterization of the tasks which are solvable in the presence of one faulty processor (1-solvable) was given in [BMZ]. This characterization used certain graphs associated with a given task, and by examining the properties of these graphs it is determined whether the task can be solved in the presence of one faulty processor; in case of a positive answer, an algorithm which uses traversing of these graphs was given.

Thus, determining whether a task is 1-solvable is reduced to testing certain combinatorial properties. Verifying whether these combinatorial properties hold for each of the tasks studied in the references above is relatively straightforward. This raises the question of whether there is an efficient algorithm that receives the specification of a distributed task for \(N\) processors as an input, and determines whether this task is 1-solvable. This latter question is naturally generalized to a similar question in which the number of processors failures is at most \(t\), for some integer \(t\). In this paper we show that this problem is NP-hard for any fixed positive \(t\) and \(N\), provided \(N > 2t\). For the specific case \(t=1\) we actually show that the problem is NP-complete.

It should be noted that there are distributed tasks which cannot be computed even by centralized algorithms (e.g., tasks defined by non-recursive functions). Therefore, we restrict the discussion in this paper only to finite tasks, which are tasks with finitely many possible inputs and outputs. A specification of a finite task consists of a list of legal input vectors, and a (finite) list \(G(T)\) of legal decision vectors for each input vector. We first consider the case \(t=1\), and then sketch a generalization of the NP-hardness proof to arbitrary \(t\), by reducing it to the case \(t=1\). Interestingly enough, there does not appear to be a straightforward reduction of this type, and our reduction is heavily based on the specific characterization given in [BMZ].

The NP-hardness proof is based on a polynomial time reduction from the following problem:

**Input:** A \(k\)-partite graph \(H_k\), together with a partition of the vertices of \(H_k\) into \(k\) independent sets.

**Property:** \(H_k\) contains a clique of size \(k\) (in short, a \(k\)-size clique).

This problem is NP-complete by the original reduction from the Satisfiability problem to the Maximum Clique problem in \([K]\).

The rest of the paper is organized as follows: in Section 2 we briefly review the 1-solvability characterization of [BMZ] with the necessary definitions. In Section 3 we describe a reduction from the above problem to the problem of determining whether a distributed task for \(k\) processors is 1-solvable (note that \(k\) is the number of sets partitioning the graph, and hence is not fixed in this reduction). In Section 4 we provide a proof of correctness of the reduction. In Section 5 we modify the reduction and the proof so that the resulted task is for 3 processors (independent of \(k\)). In Section 6 we show that the problem for finite tasks is NP-complete. In Section 7 we show that the \(t\)-solvability problem is NP-hard for any fixed \(t\) and \(N > t/2\), by a reduction from the 1-solvability problem.
2. THE I-SOLVABILITY CHARACTERIZATION

2.1 Asynchronous Systems

An asynchronous distributed system is composed of a set \( V = \{ P_1, P_2, \ldots, P_n \} \) of \( n \) processors \((n \geq 3)\), each having a unique identity. We assume that the identities of the processors are mutually known, and w.l.o.g. that the identity of \( P_i \) is \( i \). Our results are applicable also to the model in which the identities are not mutually known (or absent, provided that the inputs are distinct). The processors are connected by communication links, and they communicate by exchanging messages along them. A distributed protocol is a collection of algorithms run by the processors. Each such algorithm contains operations of sending/receiving messages and of local computations. An execution of a protocol is a sequence of such operations. For more detailed definitions see, e.g., [FLP]. 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 the sequel).

In this paper we consider only decision protocols, in which each processor has to decide - an action that can be considered as writing a write-once register.

2.2 Decision Tasks

Let \( X \) and \( D \) be sets of input values and decision values, respectively. A distributed decision task \( T \) for \( N \) processors \((P_1, P_2, \ldots, P_N)\) is a function

\[
T : X_T \rightarrow 2^{D_T},
\]

where \( X_T \subseteq X^N \). \( X_T \) is called the input set and \( D_T = \cup \ T(\mathcal{O}) \) is the decision set of the task \( T \). Each \( x \in X_T \)

vector \( \mathcal{x} = (x_1, x_2, \ldots, x_N) \in X_T \) 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 \( \mathcal{d} = (d_1, d_2, \ldots, d_N) \in D_T \) 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 set of allowable decision vectors.\(^2\)

Examples:

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

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

3. Order Preserving Renaming [ABDKPR]: This task is defined for a given integer \( K \), where \( K \geq n \). The input set \( X_T \) is the set of all vectors \((x_1, \ldots, x_n)\) of distinct integers. For a given input \( \mathcal{x} \in X_T \), \( T(\mathcal{O}) \) is the set of all integer vectors \( \mathcal{d} = (d_1, \ldots, d_n) \) satisfying \( 1 \leq d_i \leq K \) and such that for each \( i,j \), \( x_i < x_j \) implies \( d_i < d_j \).

\(^2\) Unlike in [BMZ], we allow here, in order to simplify the reduction, that \( T(\mathcal{O}) = \{ \mathcal{O} \} \). The result in [BMZ] holds trivially also for such tasks, since if \( T(\mathcal{O}) = \{ \mathcal{O} \} \) for some input vector \( \mathcal{x} \) then \( T \) is not I-solvable.
2.3 Faults and 1-Solvability

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

Definition: A protocol \( \alpha \) 1-solves a task \( T \) if for every execution of \( \alpha \) on input \( x \in X_T \) 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 \( T(x) \).

When such a protocol \( \alpha \) exists we say that the task \( T \) is 1-solvable.

2.4 The characterization

Let \( S \subseteq A^N \) be an arbitrary set. 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 \((\vec{S}_1, \vec{S}_2) \in E_S \) iff \( \vec{S}_1 \) and \( \vec{S}_2 \) are adjacent.

A partial vector is a vector in which one of the components is not specified; this entry is denoted by \( \bullet \). For a vector \( \vec{S} = (S_1, \ldots, S_N) \), \( \vec{S}^i \) denotes the partial vector obtained by assigning \( \bullet \) to the \( i \)-th component of \( \vec{S} \), i.e., \( \vec{S}^i = (s_1, \ldots, s_{i-1}, \bullet, s_{i+1}, \ldots, s_N) \). For a set of vectors \( S \), \( S^i = \{ \vec{S}^i : \vec{S} \in S \} \).

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 \( S \) of vectors of length \( N \). Then for each clique \( C \) in \( G(S) \) there exists an integer \( i \), \( 1 \leq i \leq N \), such that every two 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 constitutes 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 of size at least two defines a unique partial vector \( \vec{S}^i \) 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 \( \vec{S}^i \) of a vector \( \vec{S} \) defines a unique maximal \( i \)-clique (denoted as \( C(\vec{S}^i) \)), that includes \( \vec{S} \) and all the vectors that differ from \( \vec{S} \) in the \( i \)-th component only (i.e., that agree with \( \vec{S}^i \)).

The adjacency graph \( G(X_T) \) of the input set \( X_T \) is called the input graph of \( T \). For an input vector \( \vec{x} \), \( G(T(\vec{x})) \) is the decision graph of \( T \).

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

Definition: A task \( T \) is pointwise connected if \( G(T(\vec{x})) \) is connected for each \( \vec{x} \in X_T \).

Definition: Let \( C(\vec{d}^i) \) be a maximal \( i \)-clique in \( G(X_T) \). A partial vector \( \vec{d}^i \) is a covering vector for \( C(\vec{d}^i) \) in \( T \) if for each input vector \( \vec{x} \) in \( C(\vec{d}^i) \) there is an extension of \( \vec{d}^i \) to a decision vector \( \vec{d} \) in \( T(\vec{x}) \).
Theorem 1: ([BMZ]) A task \( T \) is 1-solvable if and only if there exists a restriction \( T' \) of \( T \) satisfying the following:

(1a) \( T' \) is pointwise connected, and

(1b) For each \( i, 1 \leq i \leq N \), for each maximal \( i \)-clique \( C(\ell^i) \) in \( G(X_T) \) there is a covering vector \( \ell^i_j \) in \( T' \); moreover, there is a (centralized) algorithm that on input \( \ell^i \) outputs such \( \ell^i_j \).

3. THE REDUCTION

Let \( H_k = (V,E) \) be a \( k \)-partite graph, where the \( k \) sets partitioning \( V \) are \( V_1, \ldots, V_k \). We present a polynomial time reduction of \( H_k \) to a task \( T \) which is 1-solvable iff \( H_k \) contains a \( k \)-size clique. The reader is advised to follow with the example in Figure 1.

The set \( D \) of input values is the set of integers in \( \{1,\ldots,k\} \). The set \( D \) of decision values is the set \( \mathcal{S} \) of the edges of \( H_k \).

The task \( T \) is defined for \( k \) processors. First, we define the input set and the input graph of \( T \), then we define the decision set of \( T \), and finally we define the task \( T \) as a mapping from its input vectors to subsets of its decision vectors.

\( X_T \), the input set of \( T \), is defined in such a way that the input graph \( G(X_T) \) is isomorphic to the \( k \)-size clique \( K_k \), in which each edge is replaced by a path of length \( k \). It contains \( m + (k-1)^2 k / 2 \) input vectors: one vertex-vector for each vertex of the \( k \)-size clique, and \( k-1 \) edge-vectors for each edge of the \( k \)-size clique.

For a constant \( c, \epsilon \) denote the constant vector \((c, \cdots, c)\). The vertex-vectors in \( X_T \) are the vectors \( x_i^j = \ell, \quad i = 1, \ldots, k \). The edge-vectors that form the path connecting \( x_i^j \) with \( x_j^j \), denoted by \( x_i^{j-1} \) and \( x_i^{j+1} \), are defined as follows: \( x_i^{j-1} \) has all its entries equal to \( j \), except the \( j \)-th entry, which is \( j \). \( x_i^{j+1} \) has all its entries equal to \( j \), except the \( j \)-th entry which is \( k \); the entries of the other \( k-3 \) vectors on this path are defined such that \( x_i^j \) is adjacent to \( x_i^{j+1} \) (for \( i = 1, \ldots, k-2 \)). Figure 2 depicts the input graph for \( k = 4 \).

For the definition of the decision set \( D_T \) we need the following:

Definition: Let \( \epsilon \) and \( d \) be two constant vectors. \( CON(\epsilon,d) \) is the set of vectors \( \{\epsilon = (\epsilon_1, \cdots, \epsilon_k), \quad (d,\epsilon_1, \cdots, \epsilon_k), \quad (d, \cdots, \epsilon_k, \epsilon_1), \quad \cdots, (d, \cdots, \epsilon_k, \epsilon_1)\} \). Note that \( G(CON(\epsilon,d)) \) is a path of length \( k \), connecting \( \epsilon \) and \( d \). For a single constant vector \( \epsilon \) we define \( CON(\epsilon) = \{\epsilon\} \). For a sequence of \( m > 2 \) distinct constants vectors \( \epsilon_1, \cdots, \epsilon_m \), we define \( CON(\epsilon_1,\cdots,\epsilon_m) = \bigcup_{\ell = 1}^{m-1} CON(\epsilon_1, \cdots, \epsilon_\ell, \epsilon_{\ell+1}) \). Thus \( G(CON(\epsilon_1,\cdots,\epsilon_m)) \) is a path of length \( (m-1)k \), starting at \( \epsilon_1 \), going in order via all the \( \epsilon_i \)'s and ending in \( \epsilon_m \).

The decision set \( D_T \) is defined as follows. For each vertex \( u \) in \( V \) with incident edges \( e_1, \cdots, e_l \), let \( S_u = CON(e_1, \cdots, e_l) \). Then \( D_T = \bigcup_{u \in V} S_u \).

Finally, the mapping \( T \) from \( X_T \) to \( D_T \) is defined as follows. For an edge-vector \( x_i^j \), \( T(x_i^j) = \{\epsilon \in \epsilon \mid \epsilon \) connects a vertex in \( V_1 \) to a vertex in \( V_j \} \). Note that \( G(T(x_i^j)) \) consists of \( m_j \) isolated vertices, where \( m_j \) is the number of edges connecting vertices from \( V_i \) and \( V_j \). For a vertex vector \( x_i^j \), \( T(x_i^j) = \bigcup_{u \in V_i} S_u \).
For each edge-vector \( t_{ij} \) in \( X_T \).

\[ T'(t_{ij}) \cap T'(t_{ij+1}) \neq \emptyset \]

Note that \( G(T'G) \) consists of \( |V_i| \) connected components, where each component is \( G(S_u) \) for some \( u \in V_i \).

4. PROOF OF CORRECTNESS

We show that a restriction \( T' \) of \( T \) satisfying Theorem 1 exists if the graph \( H_k \) contains a \( k \)-size clique.

(a) Assume first that \( T \) is \( 1 \)-solvable, and let \( T' \) be a restriction of \( T \) satisfying Theorem 1. We show that \( H_k \) contains a \( k \)-size clique.

By (1a) of Theorem 1, for each vertex-vector \( \vec{x}_i \), the graph \( G(T'(\vec{x}_i)) \) must be connected. Since each connected component of \( G(T'(\vec{x}_i)) \) is \( G(S_u) \) for some \( u \in V_i \), this means that for some vertex \( u_i \in V_i \), \( T'(\vec{x}_i) \) is included in \( S_u \). We claim that the vertices \( \{u_i : i = 1, \cdots, k\} \) form a clique in \( H_k \). For this, we must show that for each \( i, j \), there is an edge connecting \( u_i \) and \( u_j \).

Consider an edge-vector \( \vec{x}_i \). Since \( G(T'(\vec{x}_i)) \) consists of isolated vectors, each representing an edge in \( H_k \), (1a) implies that \( T'(\vec{x}_i) \) is a single constant vector \( e \), representing a single edge \( e \). Now consider vectors \( \vec{x}_i^{k-1} \) and \( \vec{x}_i^k \) that form a maximal \( l \)-clique in \( G(X_T) \) (for some \( l \)). Let \( T'(\vec{x}_i^k) = \{ e \} \) and \( T'(\vec{x}_i^{k-1}) = \{ e_1 \} \) By (1b), \( T' \) has a covering vector \( d' \) for the above input \( l \)-clique. This implies that \( e \) and \( e_1 \) may differ only in the \( j \)-th entry; and hence that \( e = e_1 \). We conclude that for \( r = 1, \cdots, k-1 \), \( T'(\vec{x}_i^r) = \{ e \} \) for some \( e \). It remains to show that the edge \( e \) connects \( u_i \) and \( u_j \).

Consider the maximal input \( l \)-clique consisting of the vectors \( \vec{x}_i^l \) and \( \vec{x}_j^l \). Again, by (1b), \( T' \) has a covering vector for that clique. Since for every constant vector \( e \) in \( D_T \) and for every \( u \in V_i \), either \( e \) is in \( S_u \) or \( e \) is not adjacent to any vector in \( S_u \), we must have that \( e \) is in \( S_u \), meaning that \( e \) is incident to \( u_i \). By a similar argument for the input clique consisting of \( \vec{x}_i^{k-1} \) and \( \vec{x}_j^k \), \( e \) is incident to \( u_j \). This completes the proof of (a).

(b) Now assume that \( H_k \) contains a \( k \)-size clique, where \( u_i \) is the vertex from \( V_i \) in this clique, and \( e_{ij} \) is the edge connecting \( u_i \) and \( u_j \).

We define a restriction \( T' \) of \( T \) satisfying Theorem 1:

For each vertex-vector \( \vec{x}_i \) in \( X_T \), \( T'(\vec{x}_i) = S_u \).

For each edge-vector \( \vec{x}_i^j \) in \( X_T \), \( T'(\vec{x}_i^j) = \{ e_{ij} \} \).

For (1a), it is straightforward to verify that for each input vector \( \vec{x}_i \), \( G(T'(\vec{x}_i)) \) is connected. For (1b), we only need to inspect maximal \( l \)-cliques of size at least two, since a maximal \( l \)-clique of size one always has a covering vector in \( T' \), provided that for each \( \vec{x}_i \), \( T'(\vec{x}_i) \neq \emptyset \). Now, every maximal \( l \)-clique (of size at least two) in \( G(X_T) \) consists of two adjacent vectors, \( \vec{x}_i \) and \( \vec{x}_j \). By the definition of \( T' \), \( T'(\vec{x}_i)^\cup T'(\vec{x}_j) \neq \emptyset \). The proof is completed by observing that, for every vector \( d \) in \( T'(\vec{x}_i)^\cup T'(\vec{x}_j) \), \( \vec{x}_i \) is a covering vector for the \( l \)-clique \( (\vec{x}_i, \vec{x}_j) \).
5. MODIFYING THE PROOF FOR A FIXED NUMBER OF PROCESSES

In this section we show how to modify the reduction and the proof to show that the problem remains NP-hard even when restricted to tasks for three processors. This modification can be easily adjusted for any fixed number $N > 3$ of processors.

The only reason for using vectors of $k$ entries in the reduction in Section 5 is to prevent adjacency relations among edge vectors $(\vec{X}_i^j; i < j)$ and $(\vec{X}_j^k; j < i)$ that are adjacent to a vertex vector $\vec{X}^i$ in $\vec{X}_i$. We now achieve the same effect by using vectors of only three entries, by converting every vertex vector to a clique of $k-1$ vectors, each being adjacent to one edge vector.

Let $H_k$ be a $k$-partite graph. We define below a task $T$ for three processors, such that $T$ is 1-solvable iff $H_k$ contains a clique of $k$ vertices. The set $X$ of input values is now the set of integers in $[1..k]$, together with the set of pairs $[i,j]$ of such integers, where $i < j$.

The input graph $G(X_T)$ is obtained from the $k$-size clique $K_k$, by replacing each vertex $i$ in it by $k-1$ vertices which are the vertex vectors $\vec{X}_i^1, \vec{X}_i^2, \cdots, \vec{X}_i^{k-1}, \vec{X}_i^{k+1}, \cdots, \vec{X}_i^k$, where $\vec{X}_i^j = ([i,j], [i,j])$ when $i < j$, otherwise $\vec{X}_i^j = ([i,j], [i,j])$ (note that these vectors form a 3-clique of $k-1$ vertices). An edge connecting vertices $i$ and $j$ (for $i < j$) in $K_k$ is replaced by the path $(\vec{X}_i^j, \vec{X}_i^j, \vec{X}_i^j)$, where the edge vector $\vec{X}_i^j = ([i,j], [i,j])$ (we now need only one edge vector on each edge compare to the $k-1$ before). Thus we now have $k(k-1)$ vertex vectors and $k(k-1)/2$ edge vectors. See Figure 3 for the input graph of $k = 4$.

The mapping $T$ is defined similarly to the definition of the mapping $T$ in section 3, that is: $T(\vec{X}_i^j) \rightarrow T(\vec{X}_i^j)$ and $T(\vec{X}_i^j) \rightarrow T(\vec{X}_i^j)$.

The correctness proof is similar to the proof in Section 4. For direction (a), it is also needed to observe that if a task $T'$ is a restriction of $T$ that satisfies Theorem 1, then for each $i$ there is a vertex $u_i \in V_i$ such that for each $j$, $T'(\vec{X}_i^j) = S_u$ (otherwise, the maximal 3-clique $(\vec{X}_i^j, \vec{X}_i^j, \vec{X}_i^j)$ will not have a covering vector in $T'$). For direction (b), we map $T'$ all the $k-1$ vertex vectors that now replacing $\vec{X}_i^j$ to $S_u$.

6. DECIDING 1-SOLVABILITY OF FINITE TASKS IS NP-COMPLETE

A finite task is a task $T$ given by a (finite) list of legal input vectors, and for each input vector $\vec{x}$ a (finite) list $T(\vec{x})$ of legal decision vectors for $\vec{x}$. In this section we show that the 1-solvability problem of finite tasks is in NP. Since the reduction that we have presented was to a finite task, it follows that this problem is NP-complete. We present here a non-deterministic polynomial algorithm for deciding whether a given finite task $T$ is 1-solvable:

(a) Define $T'$: choose a connected sub-graph from $G(T(\vec{x}))$ for each input vector $\vec{x}$, and define $T'(\vec{x})$ to include exactly the vectors in that connected sub-graph.

(b) Check that there is a covering vector in $T'$ for each maximal input $i$-clique. When the input vectors of the $i$-clique are given, this amounts to intersecting the sets $T(\vec{X}_i^j)$ (of all the vectors $\vec{x}$ in the $i$-clique). Now, finding all the cliques in a graph cannot be done in non-deterministic polynomial time, but since this is an adjacency graph - all the cliques are $i$-cliques and we can easily find all of them in deterministic $O(N^2 I^2)$ time ($I$ - the number of input vectors, $N$ - the number of processors) by scanning the list of input vectors (for each $i$, for each input vector $\vec{X}^i$, find the set of input vectors that agree with $\vec{X}^i$).
7. DECIDING $\tau$-SOLVABILITY IS NP-HARD FOR ANY FIXED $\tau$

In this section we reduce the 1-solvability problem to a $\tau$-solvability problem, for any fix $\tau$, $1 < \tau < N/2$. Given a task $T$ for $N$ processors, we construct a task $T'$ for $N$ processors which is solvable in the presence of $\tau$ faulty processors if $T$ is solvable in the presence of one faulty processor. The reader is advised to follow with the example in Figure 4.

**Definition:** For vector $x = (x_1, x_2, \ldots, x_N)$, $M(x)$ is a vector in which each entry is in a $\tau$-tuple: $M(x) = ((x_1, x_2, \ldots, x_\tau), \ldots, (x_{N-\tau+1}, x_{N-\tau+2}, \ldots, x_N))$.

The input set of $T$ is $X_T = \{M(x) : x \in X_T\}$.

Let $d_1$, $d_2$ be adjacent vectors. Then the Hamming distance between $M(d_1)$ and $M(d_2)$ is $\tau$. We define $\text{CONN}(M(d_1), M(d_2))$ to be the set of vectors $y_1, \ldots, y_{\tau-1}$ as follows: $y_i$ is equal to $M(d_1)$ with one exception, which is the smallest entry in which $M(d_1)$ and $M(d_2)$ differ: this entry in $y_i$ is equal to that of $M(d_2)$. For $2 \leq i \leq \tau - 1$, $y_i$ is adjacent to $y_{i-1}$, and $y_{\tau-1}$ is adjacent to $M(d_2)$ - see Figure 4.

The mapping $T$ is defined as follows: $T(M(x)) = \{M(d) : d \in T(x)\} \cup \{\text{CONN}(M(d_1), M(d_2)) : d_1 < d_2\}$. (lexicographiclly) are adjacent vectors in $T(x)$).

**Correctness proof:**

We prove below that $T$ is $\tau$-solvable iff $T$ is 1-solvable. We start with the following proposition.

**Proposition:** There is a path between $M(d_1)$ and $M(d_2)$ in $G(T(M(x)))$ if and only if there is path between $d_1$ and $d_2$ in $G(T(x))$.

**Proof:** The "if" part follows directly from the construction.

For the "only if" part, consider a path $p$ in $G(T(M(x)))$, starting from $M(d_1)$. Any vertex in $p$ is either $M(d)$ for a unique $d \in T(x)$, or is in $\text{CONN}(M(d), M(d'))$ for some unique pair of vertices $d$, $d'$ which are adjacent in $T(x)$. It is sufficient to prove that in these cases $d$ and $d'$ must be in the same connected component of $d_1$. This is proved by induction on the length of $p$, using the following observation whose proof is left to the reader.

Let $\overline{x}$, $\overline{y}$ be two vertices in $\overline{G(T(M(x)))}$. Then $\overline{x}$, $\overline{y}$ are adjacent iff either:

(a) there are $\overline{b}$, $\overline{c}$ which are adjacent in $\overline{G(T(x))}$, and $\overline{x}$, $\overline{y}$ are adjacent on the path consisting of $M(\overline{b})$, $M(\overline{c})$ and $\text{CONN}(M(\overline{b}), M(\overline{c}))$, or

(b) there are three vectors $\overline{b}$, $\overline{c}$, $\overline{d}$ in $\overline{T(x)}$ which form an $i$-clique for some $i$, and $\overline{x}$ and $\overline{y}$ are adjacent to $M(\overline{b})$, and belong to $\text{CONN}(M(\overline{b}), M(\overline{c}))$ and $\text{CONN}(M(\overline{b}), M(\overline{d}))$, resp. □

We now assume that $T$ is 1-solvable, and prove that $T$ is $\tau$-solvable. Let $T'$ be a restriction of $T$ satisfying Theorem 1. The proof is by constructing a protocol that $\tau$-solves $T$. This protocol is similar to that of [BMZ], and we present here only the necessary modifications. The main difference in the protocols is that in this case, in any round of the protocol each processor waits until it receives $N-\tau$ messages, and not $N-1$ as in [BMZ]. The construction is based on the fact that $N-\tau$ elements of the input vector $M(x)$ determine at least $N-1$ elements of $x$. A processor who knows after phase $B$ of the protocol only $x'$ decides on $M(\text{COMP.COVER}(G'))$, where COMP.COVER is the same procedure defined in [BMZ] for $T'$. In phase C of the protocol in [BMZ], the processors converge to two adjacent vectors in an anchors tree in $G(T(x))$; the protocol here similarly converges to two adjacent vectors in $G(T(M(x)))$, where each original $i$-anchor $A_i$ is replaced by the $i$-anchor $M(A_i)$. The existence of such anchors
tree is guaranteed by the existence of such a tree in $G(T'(x))$ and by the proposition above.

For the other direction, assume that $T$ is $t$-solvable, and let $a$ be a protocol that $t$-solves it. We define a restriction $T'$ of $T$ satisfying Theorem [BMZ]. Let $D_a(M(t))$ denote the set of all vectors output by $a$ in all the complete executions (in which all the processors decide) on input $M(t)$. Since $a$ also $i$-solves $T$, from [BMZ] we have that $G(D_a(M(t)))$ is connected, and of course $D_a(M(t)) \subseteq T(M(t))$. We define $T'(x) = \{ \vec{d} \mid M(t) \subseteq T(M(t)) \}$. From the connected component in $G(T(M(t)))$ which contains $G(D_a(M(t))))$.

By the proposition, $G(T'(x))$ is connected, for each input vector $x$. We have to show that for each maximal input clique $C(x)$, there is a covering vector $\vec{d}^i$ in $T'$. To see this, note that the only entries of $M(t)$ that contain $x_i$ are in positions $i-t+1$ through $i$ of this vector. Consider an execution of $a$ in which the corresponding $t$ processors $P_{i-t+1}, P_{i-t+2}, \ldots, P_i$ delayed from the start until all others decide, and the input to a non-delayed processor $P_j$ is the $j$-th entry of the vector $M(t)$, i.e., it is the $t$-tuple $(x_{j-1}, \ldots, x_{i-t+1})$. Let $\vec{d}^i$ be the partial decision vector output by the non-delayed processors $(N-t$ entries of $\vec{d}^i$ are $t$-tuples, and the $t$ entries corresponding to the delayed processors are $\ast$). $\vec{d}^i$ must be extendible to a legal decision vector in $T'$.

$\vec{d}^i$, the covering vector for $C(x)$, is defined as follows: (a) if $\vec{d}^i$ is extendible to a vector $M(t)$ for some $x$, then let $\vec{d}^i$ be $Y_i$; (b) otherwise, $\vec{d}^i$ is extendible to a vector in $CONN(M(t), M(t))$, for some adjacent $Y_1, Y_2$; let $\vec{d}^i$ be $Y_1$.

We now prove that $\vec{d}^i$ is indeed a covering vector for $C(x)$. For each extension of $\vec{d}^i$ to a legal input vector $x$ in $X$, consider the case where the inputs to the delayed processors in the above execution of $a$ are according to $M(t)$, and they are not delayed any more. Eventually all processors must decide, resulting a decision vector $\vec{d}$ in $T(M(t))$, which extends the partial decision vector $\vec{d}^i$. If $\vec{d}$ is $M(t)$ for some $\vec{d}^i$, then $\vec{d}^i$ was defined by (a) above, and by the definition of the mapping $M(t)$, $\vec{d}$ must be an extension of $\vec{d}^i$, and $\vec{d}$ belongs to $T'(x)$ by its definition. Otherwise, $\vec{d}$ belongs to $CONN(M(t), M(t))$, for some adjacent $\vec{d}^i$. In this case one of $\vec{d}^i$ must be an extension of $\vec{d}^i$, and they both belong to $T'(x)$ by its definition.

REFERENCES


Figure 1: The construction of $T$ from $H_k$ ($k=3$).
Figure 2: The input graph $G(X_T)$ for $k=4$. 
a clique of $k-1$ vectors instead of $x_i$ before.

Figure 3: The input graph of three processors task ($k=4$).
Figure 4: The transformation of an input vector and its decision graph from the task \( T \) to the task \( \tilde{T} \) (\( N=7 \), \( t=3 \)).