The Complexity of Almost-Optimal Coordination[*]

Rida A. Bazzi[†]
Gil Neiger

GIT–CC–93/68

November 30, 1993

Abstract

The problem of fault-tolerant coordination is fundamental in distributed computing. In the past, researchers have considered the complexity of achieving optimal simultaneous coordination under various failure assumptions. This paper studies the complexity of achieving simultaneous coordination in synchronous systems in the presence of send/receive omission failures. It had been shown earlier that achieving optimal simultaneous coordination in these systems requires NP-hard local computation. In this paper, we study *almost-optimal* coordination, which requires processors to coordinate within a constant additive or multiplicative number of rounds of the coordination time of an optimal protocol. We show that achieving almost-optimal coordination also requires NP-hard computation.

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

[*]This work was supported in part by the National Science Foundation under grants CCR-9106627 and CCR-9301454.
[†]This author was supported in part by a scholarship from the Hariri Foundation.

1 Introduction

Coordinating the activity of the processors in a distributed system is a fundamental problem in distributed computing. Such problems require processors to agree on a common action to perform and to ensure that the action chosen is valid given the context within which they are operating. This paper specifically considers the complexity of *fault-tolerant* coordination in the presence of *general omission* failures, by which faulty processors may omit to send or receive messages. Fault-tolerant coordination requires that the nonfaulty processors successfully coordinate their actions despite the failures of others. There is a large body of literature within computer science that has studied fault-tolerant coordination problems, such as *Reliable Broadcast* and *Distributed Consensus*. Fischer [6] provides a survey of many such problems.

This paper considers *simultaneous* coordination problems in synchronous systems in which algorithms operate in *rounds* of communication (simultaneous coordination is not possible in asynchronous systems). The results in this paper are based on the relationship between coordination and different forms of *processor knowledge* [9]. It is well-established that knowledge can be used to characterize and improve solutions to various problems in distributed computing [3,5,8,10–14,17]. For example, Moses and Tuttle [13] showed that *common knowledge* is necessary for the solution of simultaneous coordination problems and used this fact to derive optimal solutions to such problems. In particular, they proved that achieving optimal coordination in the presence of general omission failures requires processors to perform NP-hard local computation between rounds of communication. Neiger and Tuttle [17] subsequently showed related results considering stronger forms of coordination and of common knowledge.

It is known that many NP-complete problems can be solved *approximately* in polynomial time [7]. Perhaps almost-optimal algorithms for simultaneous coordination might require only polynomial-time local computation (in systems with general omission failures). A simultaneous coordination algorithm is *almost-optimal* if processors decide within a constant additive or multiplicative number of rounds of the decision time of an optimal algorithm. The possibility of such an algorithm in the multiplicative case is suggested by studies of translations between models of failures [1,2,4,15,18]. For example, Moses and Tuttle showed the existence of optimal algorithms that tolerate crash failures. Neiger and Toueg [15] showed how algorithms tolerant of crash failures could be converted to tolerate general omission failures by doubling the number of rounds used; local computation time was increased only by a polynomial amount. If this translation were applied to the optimal (crash-tolerant) algorithm of Moses and Tuttle, perhaps it would result in a polynomial-time algorithm that is almost-optimal within a constant factor (i.e., decision time would be at worst twice optimal).

We demonstrate that this method does not work. In particular, we show that, in any almost-optimal coordination algorithm (for general omission failures), processors may still be required to perform NP-hard local computation between rounds of communication.

The paper is organized as follows. In Section 2, we define the model of the system. In Section 3, we give a formal definition of coordination algorithms and of almost-optimal coordination. In Section 4, we introduce the necessary knowledge theoretic background needed for the development of the results. In Section 5, we present our results. Section 6 concludes the paper with a brief discussion.

2 Definitions

This section defines a model of a synchronous distributed system. This model is similar to others used to study knowledge and coordination [5,9,10,13,14,17].

A distributed system consists of a set P of n processors connected by a communication network such that any processor can send a message to any other. All processors share a clock that starts at time 0 and advances in increments of one. Computation proceeds in a sequence of *rounds*, with round r taking place between time $r-1$ and time r. At the start of a round, each processor may receive an external input. The inputs to all the processors in a round form the *input vector* for that round. In every round, a processor sends messages to other processors, receives messages that have arrived since the last round, performs some local computation and, optionally, a coordination action. At any given time, a processor's *local state* consists of the time on the global clock, the messages it has sent and received, the sequence of external inputs it has received, and the coordination actions it has performed. A *global state* is a tuple of local states, one per processor.

A message sent in a round is either received in that round or never received. Message losses are due to processor failures. We consider general omission failures, in which a faulty processor may omit to send or receive messages. Up to t processors can fail in an execution. We assume that the faulty behavior in an execution is independent of the inputs to the processors.

Processors follow a *protocol*. A protocol has two components: the *communication component* specifies the messages a processor is required to send for a round as a function of the processor's local state at the beginning of that round, and the *coordination component* (or *action function*) specifies for every round whether a processor should perform a coordination action (also function of local state). A *history* is a protocol paired with an infinite sequence of global states, one per round.

Some processors correctly follow the protocol and are thus *nonfaulty*. Other processors are *faulty*. In any round ℓ, we associate with each processor p two sets of processors $S(p,\ell)$ and $R(p,\ell)$ that are respectively the sets of processors p failed to send to or to receive from in round ℓ. A processor is faulty if and only if $S(p,\ell) \cup R(p,\ell)$ is nonempty for some ℓ. The sets $S(p,\ell)$ and $R(p,\ell)$ for all p and ℓ define the *failure pattern*.

A *run* is a history paired with a failure pattern. $\mathcal{N}(\rho)$ denotes the set of processors nonfaulty in run ρ (we will use $\mathcal{N}$ when ρ is clear from context). A failure pattern is *compatible* with a history if it accounts for all missing messages in that history. For example, assume that in some round processor p does not receive a message that it is supposed to get from processor q and that all other messages are delivered as specified by the protocol (note that this is an informal partial description of a history). It is possible that either p omitted to send or that q omitted to receive a message. Both of these failure patterns are compatible with the history just described. This work identifies a system with the set of all runs of a communication protocol in that system.

An ordered pair (ρ, ℓ), where ρ is a run and ℓ is a natural number, is called a *point* and represents the system after the first ℓ rounds of ρ. The local state of processor p at that point is denoted by $\rho_p(\ell)$. The input vectors for all the rounds of a run define the input pattern of the run.

In order to analyze systems, it is convenient to have a logical language to make statements about the system. A *fact* in this language is interpreted to be a property of points: a fact φ will be either true or false at a given point (ρ, ℓ), denoted $(\rho, \ell) \models \varphi$ and $(\rho, \ell) \not\models \varphi$, respectively. Fact φ is *valid in a system* if it is true at all points in the system; it is *valid* if it is valid in all systems. Although facts are interpreted as properties of points, it is often convenient to refer to facts that are about objects other than points (e.g., properties of runs). In general, a fact φ is a *fact about X* if fixing X determines the truth (or falsity) of φ.

3 Simultaneous Coordination Problems

A simultaneous coordination problem $\mathcal{C}$ is a set $\{a_1, \ldots, a_m\}$ of actions and an associated set $\{ok_1, \ldots, ok_m\}$ of enabling conditions that are facts about the input and the existence of failures. A protocol $\mathcal{P}$ implements a simultaneous coordination problem in a system if and only if the following holds of every run ρ of $\mathcal{P}$ in the system:

- each nonfaulty processor performs at most one action from the set of actions,
- any action performed by a nonfaulty processor is performed simultaneously by all of them, and
- an action is performed by a nonfaulty processor only if its associated enabling condition is true of ρ.

We note that our definition of simultaneous coordination problems is the same as the definition that Moses and Tuttle [13] give for *simultaneous choice problems.*

In this paper, we study coordination problems whose enabling condition are nontrivial facts about the input and the existence of failures. A fact is *nontrivial* if neither it nor its negation is valid in the system. Many natural simultaneous coordination problems have enabling conditions that are nontrivial facts about the input and the existence of failures.

In a given round r, the complexity of the local computation of a processor is measured as a function of r and the total number of processors n. Moses and Tuttle provide a justification of this choice of parameters. Since in any protocol, processors would be required to exchange messages containing their external inputs, we assume that the external inputs are polynomial in size. If processors perform only polynomial-time local computation, we say that the protocol runs in polynomial time.

Two runs of two protocols are *corresponding runs* if they have the same failure and input patterns. A protocol $\mathcal{P}$ implementing simultaneous coordination problem $\mathcal{C}$ is *optimal* if, for every protocol $\mathcal{P}'$ that implements $\mathcal{C}$ and every pair of corresponding runs of $\mathcal{P}$ and $\mathcal{P}'$, the nonfaulty processors perform an action in the run of $\mathcal{P}$ no later than they do in the run of $\mathcal{P}'$. $\mathcal{P}$ is *almost-optimal with additive constant k* if, for every protocol $\mathcal{P}'$ and every pair of corresponding runs of $\mathcal{P}$ and $\mathcal{P}'$, the nonfaulty processors perform an action in the run of $\mathcal{P}$ no more than k rounds after they do in the run of $\mathcal{P}'$. $\mathcal{P}$ is *almost-optimal with multiplicative constant k* if, for every protocol $\mathcal{P}'$ and every pair of corresponding runs of $\mathcal{P}$ and $\mathcal{P}'$, the following holds: if the nonfaulty processors perform an action in the run of $\mathcal{P}'$ in round r then they perform an action in the run of $\mathcal{P}$ no later than round kr. An action function of an almost-optimal protocol is an *almost-optimal action function.*

A protocol is a *full-information* protocol if processors exchange all the information they have about the run in every round; that is, in every round every processor sends its state to every other processor. Moses and Tuttle proved that, if there exists a protocol $\mathcal{P}$ that implements a simultaneous coordination problem and runs in polynomial time, then there exists a full-information protocol $\mathcal{P}'$ that implements the same problem and runs in polynomial time such that processors decide in $\mathcal{P}'$ no later than they do in $\mathcal{P}$. Therefore, in what follows, we can restrict our study to full-information protocols.

4 Knowledge and Coordination

Processor knowledge was first defined by Halpern and Moses [9] as follows. *Processor p knows φ at point (ρ, ℓ)*, denoted $(\rho, \ell) \models \mathsf{K}_p\varphi$, if $(\rho', \ell') \models \varphi$ for all runs ρ' of the system such that $\rho'_p(\ell') = \rho_p(\ell)$ Since the global clock is always part of a processor's local state, it follows that $(\rho, \ell) \models \mathsf{K}_p\varphi$, if $(\rho', \ell) \models \varphi$ for all runs ρ' of the system such that $\rho'_p(\ell) = \rho_p(\ell)$. It is useful to condition a processor's knowledge on its being nonfaulty. We say that *processor p believes φ* if p knows that, if it is in $\mathcal{N}$, φ is true. That is, $\mathsf{B}_p\varphi \equiv \mathsf{K}_p(p \in \mathcal{N} \Rightarrow \varphi)$. It is easy to see that $(\rho, \ell) \models \mathsf{B}_p\varphi$ if $(\rho', \ell) \models \varphi$ for all runs ρ' such that $r'_p(\ell) = r_p(\ell)$ and $p \in \mathcal{N}(r')$.

Because this paper deals with coordination among the nonfaulty processors, we are specifically interested in the knowledge possessed by the set $\mathcal{N}$ of nonfaulty processors. *Everyone in $\mathcal{N}$ knows φ*, denoted $\mathsf{E}\varphi$, is defined as $\bigwedge_{p \in \mathcal{N}} \mathsf{B}_p\varphi$. (Moses and Tuttle [13] and Neiger and Tuttle [17] explain why belief, and not knowledge, is appropriate for the problems considered here.) Fact φ is *common knowledge*, denoted $\mathsf{C}\varphi$, if $\bigwedge_{i \geq 1} \mathsf{E}^i\varphi$.

A useful tool for reasoning about knowledge is the *similarity graph.* The nodes of this graph are the points of the system, and there is an edge between points (ρ, ℓ) and (ρ', ℓ) if and only if there exists a processor that is nonfaulty at both points and has the same local state at both points. It is not hard to see that, if φ is true at all points adjacent to (ρ, ℓ), then $(\rho, \ell) \models \mathsf{E}\varphi$. We define the *similarity relation* $\sim$ on the similarity graph to be the transitive closure of the adjacency relation. We say that two points (ρ, ℓ) and (ρ', ℓ) are *similar* if $(\rho, \ell) \sim (\rho', \ell)$. It is not hard to see that a fact is common knowledge at a point (ρ, ℓ) if and only if it holds at all points that are similar to it [13].

One way to prove that a fact is common knowledge is to use the *induction rule* for common knowledge [9]. The induction rule says that, if $\varphi \Rightarrow E\varphi$ is valid in a system, then so is $\varphi \Rightarrow \mathsf{C}\varphi$.

Common knowledge is important for reasoning about simultaneous choice problems because, when processors perform a simultaneous coordination action, they must have common knowledge of the enabling condition of the action being performed [5,13]. In particular, processors running an optimal protocol perform a simultaneous coordination action as soon as some enabling condition becomes common knowledge. An almost-optimal action function with additive constant k requires processors to take some action in round r when some enabling condition has been common knowledge since round $r-k$. An almost-optimal action function with multiplicative constant k requires processors to take some action in round r when some enabling condition has been common knowledge since round $\lfloor r/k \rfloor$.

5 Complexity Results

In this section, we study the complexity of almost-optimal simultaneous coordination. Moses and Tuttle [13] proved that processors running an optimal protocol may be required to perform NP-hard local computation between rounds of communication. We show that processors running an almost-optimal protocol would still be required to perform NP-hard local computation between rounds of communication. Our proof is similar to that of Moses and Tuttle.

We prove the NP-hardness by giving a Turing reduction from the *Vertex Cover* problem. This is the problem of determining whether a given graph $G = (V, E)$ has a subset C of vertices (vertex cover) of size less than a given integer m such that any edge of G is incident to some vertex in C. The Vertex Cover problem is known to be NP-hard [7]. The intuition behind the reduction is that the faulty processors form a vertex cover of the graph defined by the missing messages in any round.

The following is an algorithm for Vertex Cover. It returns *yes* if and only if G has a vertex cover of size less than m; it returns *no* otherwise.

```
$c = |V|$
while $SmallerCover(G, c)$
    $c = c - 1$
if $c < m$ then
    return($yes$)
else
    return($no$)
```

SmallerCover is a boolean function that takes as input a graph G that has a vertex cover of size c and returns *true* if and only if G has a vertex cover of size less than c. We will show that, if there is an almost-optimal (within a constant additive factor) coordination function that runs in polynomial time, we would be able to implement *SmallerCover* in polynomial time, and this would then give a polynomial-time algorithm for Vertex Cover. This suffices to prove that almost-optimal coordination is NP-hard.

Let G be a graph with v vertices, known to have a vertex cover of size c. We consider a system of $n = v + 2c + 3$ processors, $t = c + 2$ of which could be faulty. We identify three processors p, q and r. A set P_G containing v other processors corresponds to the vertices of G and the remaining $2c$ processors we call A. We will consider a choice problem that has a unique action a and whose enabling condition is $\varphi =$ "p's input in round 1 is 0". (Our results can be extended to general nontrivial facts about the input and the existence of failures using standard techniques.)

Consider the following history H of a full information protocol.

- In round 1, p's external input is 0. No other input is received in H.
- In round 1, q receives p's message, but no other processor does so (this implies that p is faulty in all runs compatible with H). No processor receives any message from q in round 1 (this implies that q is faulty). There is no communication between processors p_x and p_y in P_G if and only if there is an edge between vertices x and y in G. All other messages are delivered in round 1.
- In round 2, no processor receives any message from p, while only r receives q's message. All other messages are delivered in round 2.
- After round 2, no processor ever receives a message from either p or q while all messages from all other processors are delivered.

Remember that H corresponds to a set of runs of the protocol. More than one failure pattern of the system can be compatible with it. By fixing the failure pattern, we identify a unique run. Note that φ is not known by processors in A at the end of round 2. Since $|A| = 2t > t$, at least one processor in A is correct, so φ cannot be common knowledge at the end of round 2.

In what follows we will need the following lemma of Moses and Tuttle [13, Lemma 11].

Lemma 1 (Moses and Tuttle): *Let ρ and ρ' be runs differing only in the (faulty) behavior displayed by processor p after time k, and suppose that no more than f processors fail in either ρ or ρ'. If $\ell - k \leq t + 1 - f$, then $(\rho, \ell) \sim (\rho', \ell)$.*

The following lemma applies to runs that are compatible with H:

Lemma 2: *For all k $(0 \leq k \leq c)$, φ is common knowledge at the end of round $k+3$ if and only if G has no vertex cover of size less than $c - k$.*

Proof: Assume that G has a vertex cover C of size less than $c - k$. Let ρ be a run compatible with H such that fewer than $(c-k)+2 = t-k$ processors fail by the end of the second round; such a run exists because all missing messages can be accounted for by assuming that p and q are faulty as are the (fewer than $c-k$) processors in P_G corresponding to C. Now, consider another run ρ' that differs from ρ only in the behavior of r, which is silent in ρ' after the second round. Note that at most $(c-k)+2 = t-k$ processors fail in ρ' (one more than in ρ).

Let $f = t - k$. By Lemma 1, $(\rho, \ell) \sim (\rho', \ell)$ if $\ell - 2 \leq t + 1 - f$ (since ρ and ρ' differ only in the behavior of r after time 2). Substituting for t, f and ℓ, $(\rho, k+3) \sim (\rho', k+3)$ if $k + 3 - 2 \leq t + 1 - (t-k)$ or $k+1 \leq k+1$; thus, $(\rho, k+3) \sim (\rho', k+3)$. Notice that, at point $(\rho', k+3)$, no correct processor knows φ. It is not hard to see, then, that $(\rho, k+3) \models \neg\mathsf{C}\varphi$.

Assume now that G has no vertex cover of size less than $c-k$. This means that there must be at least $c-k$ faulty processors in P_G. Let α be "p and q are faulty and P_G contains at least $c-k$ faulty processors." It follows that $\mathsf{K}_r(\varphi \wedge \alpha)$ at the end of the second round of any run compatible with H. This is because, at the end of the second round of any such run, r's state contains the information about all the messages exchanged in the first round. The missing messages cannot be accounted for unless α holds.

The remainder of the proof makes use of the idea of a processor learning a fact through a chain of distinct processors. Let $\sigma = \langle p_1, p_2, \ldots, p_d \rangle$ be a sequence of d distinct processors. We say that *processor* p_{d+1} *learns* ψ *through* σ *at time* $\ell + d$ if $\mathsf{K}_{p_1}\psi$ at time ℓ and the communication between p_i and p_{i+1} is successful in round $\ell + i$ for all i, $1 \leq i \leq d$ (p_{d+1} could be in σ). Note that, if s learns ψ through σ at time $\ell + d$ then, at time $\ell + d$, $\mathsf{K}_p(p$ learns ψ through σ at time $\ell + d)$.

Let $P_F = P_G \cup \{p, q\}$ and let $P_C = P - P_F = A \cup \{r\}$. Let β be "there exists a processor that learns $\varphi \wedge \alpha$ at time $k+3$ through a chain of $k+1$ distinct processors in P_C." Clearly, $\beta \Rightarrow \varphi \wedge \alpha$ is valid. We will show that β is common knowledge at the end of round $k+3$ of any run compatible with H. It follows from this that φ is common knowledge at such points as well.

Let ρ be a run compatible with H. The proof will use the induction rule for common knowledge: we will first show that $(\rho, k+3) \models \beta$ and then show that $\beta \Rightarrow \mathsf{E}\beta$ is valid in the system. This will imply that $\beta \Rightarrow \mathsf{C}\beta$ is valid in the system and that $(\rho, k+3) \models \mathsf{C}\beta$. We first show that $(\rho, k+3) \models \beta$. As noted above, $(\rho, 2) \models \mathsf{K}_r(\varphi \wedge \alpha)$. Since there are at least $(c-k)+2 = t-k$ faulty processors in P_F, there are at most k faulty processors in P_C. Since $|P_C| = 2c+1$, there are at least $2c+1-k$ that are correct and, since $k \leq c$, this number is at least $c+1$, which is greater than k. Let σ be r followed by any k of these correct processors; it is clear that all correct processors learn $\varphi \wedge \alpha$ through σ at time $k+3$, so $(\rho, k+3) \models \beta$.

We next show that, for any point (ρ', ℓ) such that $(\rho', \ell) \models \beta$ holds, $(\rho', \ell) \models \mathsf{E}\beta$ also holds. Let (ρ', ℓ) be such a point. Since $(\rho', \ell) \models \beta$ it follows that $(\rho', \ell) \models \alpha$, which means that there are at least $c-k+2 = t-k$ faulty processors in P_F in ρ'. Since β holds in ρ', some processor learns $\varphi \wedge \alpha$ through a chain σ of $k+1$ processors in P_C in ρ'. Since P_C contains at most $t-(t-k) = k$ faulty processors, one of the processors in σ is correct. Let p_i be this correct processor, where i is the index of p_i in σ and i is maximal among the correct processors in σ. This means that there are at least $k+1-i$ faulty processors in σ and P_C contains at most $i-1$ faulty processors outside σ. P_C contains at least $2c+1-(k+1) \geq c$ processors outside σ, at least $c-(i-1) = c+1-i$ of which are correct. It follows now that every correct processor learns $\varphi \wedge \alpha$ through a chain $p'_1, p'_1, \ldots, p'_{k+1}$ of $k+1$ distinct processors at time $k+3$. The chain is such that $p'_j = p_j$ for $1 \leq j \leq i$ and the remaining $k+1-i$ processors are chosen from the $c+1-i$ correct processors in P_C that were not in σ. Thus, $(\rho', \ell) \models \mathsf{E}\beta$. This means that $\beta \Rightarrow \mathsf{E}\beta$ is valid in the system and, by induction, so is $\beta \Rightarrow \mathsf{C}\beta$. Since $(\rho, k+3) \models \beta$, $(\rho, k+3) \models \mathsf{C}\beta$ and $(\rho, k+3) \models \mathsf{C}\varphi$. This completes the proof. $\square$

The construction of history H and Lemma 2 are not sufficient to implement *SmallerCover*. Consider the following. If a processor correct in some ρ compatible with H does not perform action a at time $k+3$, we can deduce that φ was not common knowledge at time 3 and thus that G has a vertex cover of size smaller than c. On the other hand, if a processor does perform a at time $k+3$, we can deduce only that φ is common knowledge at time $k+3$ and thus that G does not have a vertex cover of size less than $c-k$. The "gap" between c and $c-k$ prevents the direct implementation of *SmallerCover*. To implement *SmallerCover*, we define a new graph.

Let G^{k+1} be the union of $k+1$ different copies of G. G has a vertex cover of size m if and only if G^{k+1} has a vertex cover of size $m(k+1)$. When asked to implement *SmallerCover*(G,c), construct the system and history described above using G^{k+1} instead of G and $c(k+1)$ instead of c. Let $\mathcal{C}$ be the simultaneous choice problem described above and let $\mathcal{P}$ be an almost-optimal full-information coordination protocol with additive constant k that implements $\mathcal{C}$. Now, consider the history H of $\mathcal{P}$ at time $k+3$. If processors do not perform a by the end of round $k+3$, then φ is not common knowledge by the end of round 3. By substituting G^{k+1} for G and $c(k+1)$ for c in Lemma 2, we deduce that G^{k+1} has a vertex cover of size less than $c(k+1)$ or, equivalently, that G has a vertex cover of size less than c. If processors do perform a by the end of round $k+3$, then φ is common knowledge by the end of round $k+3$ [5,13]. Again, Lemma 2 implies that G^{k+1} has no vertex cover of size less than $c(k+1)-k$. This implies that G has no vertex cover of size less than c; if it did, G^{k+1} would have a vertex cover of size $(c-1)(k+1) = c(k+1)-(k+1) < c(k+1)-k$. Thus, processors perform a by the end of round $k+3$ if and only if G has no vertex cover of size less than c; the construction of G^{k+1} and Lemma 2 are sufficient to implement *SmallerCover*. This completes the proof for the case of almost-optimal protocols with additive constants.

To prove that the results holds for almost-optimal protocols with multiplicative constants, we note the following. Let ρ be a run of the history H of an almost-optimal full-information protocol with a multiplicative constant k. Processors perform a by the end of round $3k$ if φ is common knowledge by the end of round $3k$. Similarly, a processor does not perform a at the end of round $3k$ only if φ is not common knowledge at the end of round 3. The rest of the proof is almost identical to the one given above.

As noted above, our results can be extended to general nontrivial facts about the input and the existence of failures using standard techniques.

6 Conclusions

The results of this paper extend those shown earlier by Moses and Tuttle [13]. They can also be applied to the consistent simultaneous choice problems of Neiger and Tuttle [17]; solutions to such problems require that faulty processors, if they act, do so consistently and simultaneous with the nonfaulty ones. They show that the earlier NP-hardness results, which applied to optimal solutions in systems with general omission failures, also hold for almost-optimal solutions.

One conclusion that can be drawn from these results is that translations that increase fault-tolerance cannot be used to obtain almost-optimal algorithms. Consider, for example, systems with crash failures. Moses and Tuttle showed that polynomial-time optimal solutions exist for such systems. Neiger and Toueg [16] showed how crash-tolerant algorithms could be converted to tolerate general omission failures by only doubling the number of rounds used and with only a polynomial increase in local computation. The results of this paper show that, if P $\neq$ NP, then the translation of Neiger and Toueg does not preserve optimality (this could probably be shown independent of the relation between P and NP but is beyond the scope of this paper).

Note that the results hold only for constant additive and multiplicative factors. It is not hard to show that any simultaneous coordination problem has a solution that is almost optimal with an additive factor of $t-1$, where t is the number of faulty processors. The proof of Lemma 2 holds only for additive factor k, where $k \leq c = t-2$. Thus, our results and the above fact are not contradictory.

References

[1] Rida Bazzi and Gil Neiger. Optimally simulating crash failures in a Byzantine environment. In S. Toueg, P. G. Spirakis, and L. Kirousis, editors, *Proceedings of the Fifth International Workshop on Distributed Algorithms*, volume 579 of *Lecture Notes on Computer Science*, pages 108–128. Springer-Verlag, October 1991.

[2] Rida A. Bazzi and Gil Neiger. Simulating crash failures with many faulty processors. In A. Segall and S. Zaks, editors, *Proceedings of the Sixth International Workshop on Distributed Algorithms*, number 647 in Lecture Notes on Computer Science, pages 166–184. Springer-Verlag, November 1992.

[3] Piotr Berman and Juan A. Garay. Cloture votes: $n/4$-resilient, polynomial time distributed consensus in $t+1$ rounds. *Mathematical Systems Theory*, 26(1):3–20, 1993.

[4] Brian A. Coan. A compiler that increases the fault-tolerance of asynchronous protocols. *IEEE Transactions on Computers*, 37(12):1541–1553, December 1988.

[5] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a Byzantine environment: Crash failures. *Information and Computation*, 88(2):156–186, October 1990.

[6] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In M. Karpinsky, editor, *Foundations of Computation Theory*, volume 158 of *Lecture Notes on Computer Science*, pages 127–140. Springer-Verlag, 1983.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, New York, 1979.

[8] Vassos Hadzilacos. A knowledge theoretic analysis of atomic commitment protocols. In *Proceedings of the Sixth Symposium on Principles of Database Systems*, pages 129–134. ACM Press, March 1987.

[9] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, July 1990.

[10] Joseph Y. Halpern, Yoram Moses, and Orli Waarts. A characterization of eventual Byzantine agreement. In *Proceedings of the Ninth ACM Symposium on Principles of Distributed Computing*, pages 333–346. ACM Press, August 1990.

[11] Murray S. Mazer. A knowledge theoretic account of recovery in distributed systems: The case of negotiated commitment. In Moshe Y. Vardi, editor, *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 309–324. Morgan-Kaufmann, March 1988.

[12] Ruben Michel. *Knowledge in Distributed Byzantine Environments.* Ph.D. dissertation, Yale University, December 1989.

[13] Yoram Moses and Mark R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3(1):121–169, 1988.

[14] Gil Neiger and Rida Bazzi. Using knowledge to optimally achieve coordination in distributed systems. In Yoram Moses, editor, *Proceedings of the Fourth Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 43–59. Morgan-Kaufmann, March 1992.

[15] Gil Neiger and Sam Toueg. Automatically increasing the fault-tolerance of distributed algorithms. *Journal of Algorithms*, 11(3):374–419, September 1990.

[16] Gil Neiger and Sam Toueg. Simulating synchronized clocks and common knowledge in distributed systems. *Journal of the ACM*, 40(2):334–367, April 1993.

[17] Gil Neiger and Mark R. Tuttle. Common knowledge and consistent simultaneous coordination. *Distributed Computing*, 6(3):181–192, April 1993.

[18] T. K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.