Transforming Worst-case Optimal Solutions for Simultaneous Tasks into All-case Optimal Solutions

Maurice P. Herlihy
Brown University
mph@cs.brown.edu

Yoram Moses
Technion
moses@ee.technion.ac.il

Mark R. Tuttle
Intel
tuttle@acm.org

January 1, 2013

Abstract

Decision tasks require that nonfaulty processes make decisions based on their input values. Simultaneous decision tasks require that nonfaulty processes decide in the same round. Most decision tasks have known worst-case lower bounds. Most also have known *worst-case* optimal protocols that halt in the number of rounds given by the worst-case lower bound, and some have *early-stopping* protocols that can halt earlier than the worst-case lower bound (sometimes in as early as two rounds). We consider what might be called *earliest-possible* protocols for simultaneous decision tasks. We present a new technique that converts *worst-case optimal* decision protocols into *all-case optimal* simultaneous decision protocols: For every behavior of the adversary, the all-case optimal protocol decides as soon as *any* protocol can decide in a run with the same adversarial behavior. Examples to which this can be applied include set consensus, condition-based consensus, renaming and order-preserving renaming. Some of these tasks can be solved significantly faster than the classical simultaneous consensus task. A byproduct of the analysis is a proof that improving on the worst-case bound for any simultaneous task by even a single round is as hard as reaching simultaneous consensus.

1 Introduction

Decision tasks are fundamental problems in distributed computation. Each nonfaulty process begins with an input value and chooses an output value (decides), subject to conditions given by the task, even if t of the n processes fail by crashing. Some famous examples of decision tasks are

- *consensus* [PSL80, LSP82, FL81, DS82, FLP85]
- *k-set agreement* [HS99, BG93, SZ00, CHLT00]
- *condition-based consensus* [MRR03, MRR06]
- *renaming* [ABND$^+$90, HS99, AR02, HT90]

Simultaneous decision tasks [DM90, MT88, MM08] are decision tasks in which all nonfaulty processes decide in the same round. Simultaneity is important when processes must coordinate their behavior in time. Such coordination may be needed, for example, to cleanly end the execution of one protocol or one protocol phase and begin the next. Indeed, in many protocols the behavior of processes is a function of the round number, which depends on a simultaneous start.

Most decision tasks have known worst-case lower bounds, and most have known optimal protocols matching these lower bounds. Some protocols are *worst-case optimal* in the sense that every execution halts in the number of rounds given by the worst-case lower bound. Some are *early stopping* in the sense that they may occasionally halt earlier than this worst-case lower bound, sometimes as early as two rounds. Some are *all-case optimal* in the sense that, in every execution (and not just the worst-case execution), no protocol stops faster: For every behavior of the adversary (controlling input values and process failures), the all-case optimal protocol halts as early as any other protocol would in an execution with the same adversarial behavior.

Among our results, two stand out. Given a decision task P with a tight worst-case lower bound L,

1. Any worst-case optimal protocol for P can be transformed into an all-case optimal protocol for the simultaneous version of P.

2. Beating the worst-case lower bound is as hard as solving consensus: For every behavior of the adversary, if some simultaneous solution to P decides at time $k < L$ in this behavior, then simultaneous consensus can be obtained at time k in this behavior as well. In fact, the key to deciding early is agreeing that the adversary did not behave in the worst-case manner.

The novelty of our work is, after a decade of distillation, an elegant, almost too-simple combination of known results from knowledge and topology yielding powerful results and new insights.

With topology, we have an extremely powerful tool for proving lower bounds for decision tasks. In this approach, a protocol is modeled as a combinatorial structure called a *simplical complex* that describes the final states of the protocol and how much any two final states have in common (that is, which processes find the two states indistinguishable). A task, too, is modeled as a simplical complex, and a protocol solves a task if and only if a certain map exists from the protocol complex to the task complex. Lower bounds can be derived by comparing these complexes' degrees of *connectivity*, the dimensions below which their "surfaces" have no "holes." Nearly every lower bound proof for a decision task can be understood in terms of topology. In fact, for set agreement and renaming, the only lower bound proofs known to date are either based on or inspired by topological arguments.

With knowledge, we have the dominant tool for reasoning about simultaneity. A process *knows* that a predicate φ holds if φ holds in every global state compatible with the local state of the process. *Common knowledge* of φ occurs when each process knows φ, each process knows that each process knows φ, and so on. In this approach, lower bounds can be derived by observing that certain tasks require attaining common knowledge of particular facts [HM90, FHMV95], and protocol design can reduce to implementing tests for common knowledge. For simultaneous consensus, for example, Moses and Dwork [DM90] show that the optimal simultaneous consensus protocol takes time $t + 1 - W$, where W, the *waste* of the execution, is a measure of how far the execution deviates from the worst-case failure pattern (defined below).

As Moses and Raynal observed [MR08], simultaneous decision tasks require that processes agree on two things:

- on mutually-compatible decision values, and
- on a common decision round.

They started with two known results: that simultaneous consensus is solvable in $t+1-W$ rounds; and that *condition-based consensus*, which restricts the set of input vectors to those satisfying a condition involving d, is solvable in $t + 1 - d$ rounds. The question that they considered was whether it is possible to compound the savings—one based on the input structure and the other (the waste) stemming from the failure pattern. They showed that there is no "double discount"—that it is possible to solve simultaneous condition-based consensus in the minimum of $t+1-W$ and $t+1-d$ rounds, but no earlier. Moreover, the protocol that achieves this is all-case optimal: for every behavior of the adversary, determining the inputs and the failure pattern, no other simultaneous-decision protocol can decide earlier than their protocol does. This all-case optimality property is the same as that of the simultaneous consensus protocol of Dwork and Moses, which decides in $t+1-W$ rounds when the inputs are unconstrained.

Our work was inspired by [MR08], and shows that it is a particular case of a general phenomenon. A large class of simultaneous tasks can be solved in an all-case optimal fashion. Indeed, given a protocol that solves a simultaneous task in time that matches the worst-case lower bound L, we show how to use this protocol in order to obtain one that is all-case optimal, by executing it concurrently with a *continuous consensus* protocol, and deciding at the earlier of the *a priori* worst-case bound L and time $t+1-W$.

Returning to simultaneous decision tasks, common knowledge has the property that a fact about the initial state (that is, about the input values) becomes common knowledge to all processes at the same time. A decision task can usually be solved simultaneously once some fact (enough facts) about the initial state become common knowledge. On the other hand, many decision tasks are much easier than testing for common knowledge. It may take as many as $t+1$ rounds for any fact about the initial state to become common knowledge in the worst case, but the renaming task can be solved in $\log n$ rounds in the worst case. Of course, worst-case executions are the longest executions, and it is usually possible to decide much earlier in other executions. In this paper we show that, for any decision task, deciding simultaneously at any point before the decision task's worst-case lower bound—a bound typically proven using topology—reduces to testing for common knowledge, and we show that running a protocol matching the topological lower bound in parallel with a knowledge-theoretic protocol results in a protocol that is optimal in every execution (and not just the worst-case execution).

More precisely, our main results are the following:

- We show that a protocol for *continuous consensus* called ConCon [MM08], derived by knowledge-theoretic means, can be adapted to solve any decision task simultaneously in $t+1-W$ rounds, which is the time required to solve simultaneous consensus.

- We show that any protocol for a decision task can be transformed into a simultaneous protocol, and that running it in parallel with ConCon yields a simultaneous protocol that decides in time that is the minimum of original protocol's worst-case execution time and ConCon's $t+1-W$.

- We show that — for problems with tight worst-case bounds — running

> ConCon in parallel with a protocol that is optimal in the worst-case execution yields a protocol that is *all-case* optimal: For every behavior of an adversarial scheduler, our protocol halts at least as early as any other protocol for the problem would in the context of this behavior.

Interestingly, beating the worst-case bound of a simultaneous decision task is as hard as simultaneous consensus: while some simultaneous decision tasks are easy and some are hard, the cost of beating a problem's worst-case lower bound is the same $t + 1 - W$ for all problems, easy or hard.

This paper is organized as follows. Section 2 presents the synchronous model and the class of simultaneous decision tasks. It then reviews material about continuous consensus and the ConCon protocol. Section 3 considers how protocols solving a given simultaneous decision task can be composed. Section 4 presents our main theorem, showing how ConCon can be used to obtain simultaneous solutions that are optimal in all runs. Applications of this theorem are presented and discussed in Section 5. Finally, in Section 6 we discuss the results, focusing on the interaction between combinatorial topology and common knowledge that they demonstrate.

2 Model and Preliminaries

2.1 Synchronous computation

Our model of computation is a synchronous, message-passing model with crash failures. A system has $n \geq 2$ processes denoted by $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$. Each pair of processes is connected by a two-way communication link, and each message is tagged with the identity of the sender. They share a discrete global clock that starts out at time 0 and advances by increments of one. Communication in the system proceeds in a sequence of *rounds*, with round $k+1$ taking place between time k and time $k+1$. Each process starts in some *initial state* at time 0, usually with an *input value* of some kind. In every round, each process first sends a set of messages to other processes, then receives messages sent to it by other processes during the same round, and then performs some local computation based on the messages it has received.

A faulty process fails by *crashing* in some round $k \geq 1$. It behaves correctly in the first $k-1$ rounds and sends no messages from round $k+1$ on. During its crashing round k, the process may succeed in sending messages on an arbitrary subset of its links. We assume that at most $t \leq n-1$ processes fail in any given execution.

A *failure pattern* describes how processes fail in an execution. It is a graph where a vertex is a process-time pair $\langle p, k\rangle$ denoting process p and time k, and an edge is of the form $(\langle p, k-1\rangle, \langle q, k\rangle)$ denoting the fact that p succeeded in sending a message to q during round k. We write $\mathsf{Fails}(t)$ to denote the set of failure patterns in which at most t processes fail.

An *input vector* describes what input the processes receive in an execution. It is a vector $(i_1, \ldots, i_n)$ where i_k is the input to p_k.

A *run* is a description of an infinite behavior of the system. Given a run r and a time k, we write $r(k)$ to denote the global state at time k in r, and $r_p(k)$ to denote the local state of process p at time k in r.

A *protocol* describes what messages a process sends and how a process changes state during a round as a function of its local state at the start of a round and the messages received during a round. We assume that a protocol A has access to the values of n and t, typically passed to A as parameters. A run r of a protocol is uniquely determined by the protocol A, the input vector $\vec{\imath}$, and the failure pattern F, and we write $r = A[\vec{\imath}, F]$.

2.2 Decision tasks

A *decision task* is given by a relation from input vectors to output vectors. We think of each process as having an input and output register, and we think of a process as deciding on a value when it writes the value to its output register. Let I_i and O_i be sets of input values and output values for process p_i for each $i = 1, \dots, n$. Let $\mathcal{I}$ be a subset of input vectors $I_1 \times \cdots \times I_n$,[1] and let $\mathcal{O}$ be the set of output vectors $O_1 \times \cdots \times O_n$. A *decision task* P over $\mathcal{I}$ and $\mathcal{O}$ is specified by a relation on $\mathcal{I} \times \mathcal{O}$ whose projection on the first component coincides with $\mathcal{I}$. The interpretation is that if $(\vec{\imath}, \vec{o}) \in P$ and processes begin with input values in $\vec{\imath}$, then processes are allowed to decide on output values in $\vec{o}$. Formally, a protocol A solves a decision task P if every run r of A in which at most t of n processes fail satisfies the following conditions:

- *Completeness*: Every nonfaulty process decides on some output value.
- *Correctness*: The set of deciding processes choose correct values: If $\vec{\imath} \in \mathcal{I}$ is the vector of input values in the run r, then there is an output vector $\vec{o} \in \mathcal{O}$ such that $(\vec{\imath}, \vec{o}) \in P$ and each process p_j that decides in r decides on the value o_j given by $\vec{o}$.

Well-known examples of decision tasks are consensus, k-set agreement, condition-based consensus, renaming, and order-preserving renaming. We note that the values n and t are known to processes following a decision protocol in the sense that they are parameters to the protocol, and the sets $\mathcal{I}$ and $\mathcal{O}$ are known in the sense that a protocol is written with a specific family of sets (whose definitions probably depend in part on n and t) in mind.

For every decision task P, there is an associated *simultaneous decision task* denoted by $\textsc{sim}(P)$. The protocol A solves $\textsc{sim}(P)$ if, in addition to completeness and correctness, every run r of A also satisfies

- *Simultaneity*: All processes that decide in r do so in the same round.

2.3 Continuous Consensus

A central tool in our study will be the ConCon protocol [MM08], which is an efficient implementation of a *continuous consensus* service. We briefly describe continuous consensus, and then present the protocol.

[1]Most decision tasks in the literature are exhaustive, in the sense that $\mathcal{I} = I_1 \times \cdots \times I_n$. When not exhaustive, the set $\mathcal{I}$ is often called a *condition* restricting the input vectors [MRR03, MRR06].

In continuous consensus, every process maintains a copy of a "core" of information, with all copies guaranteed to be identical at all times. Moreover, this core should ultimately contain as many of the facts of interest in a given application as possible. The set of facts being "monitored" in the cores are a parameter of the service. They can involve various events such as input values, information about external events or about communication or process failures. For the purposes of this paper we focus on the case where the monitored facts are simply pairs of the form (p_j, v_j) denoting that processes p_j had input value v_j. We define a continuous consensus (CC) service with respect to initial input values to be a distributed protocol that at all times $k \geq 0$ provides each process i with a (possibly empty) core $M_i[k]$ of input values. In every run of this protocol the following properties are required to hold for all nonfaulty processes p_i and p_j:

- *Accuracy*: All values $M_i[k]$ occurred in the run.
- *Consistency*: $M_i[k] = M_j[k]$ at all times k.
- *Completeness*: If a nonfaulty process p_j has input value v_j, then $(p_j, v_j) \in M_i[k]$ must hold for some time k.

The consistency property allows for simultaneous actions that depend on input values to be consistently performed simultaneously at all times. Completeness is a liveness condition, guaranteeing that the core will contain relevant information eventually. Notice that a CC protocol can be easily used to solve simultaneous consensus, for example. If processes decide on the first input value that enters the core (and on the minimal value, in case a number of input values enter in the same round), the conditions of simultaneous consensus are all satisfied. As we shall see, a CC protocol is a useful tool for solving simultaneous decision tasks in general. In a precise sense, continuous consensus is closely related to the notion of *common knowledge*. Indeed, all of the facts in the CC core are guaranteed to be common knowledge. (We defer a more detailed definition of common knowledge and the connection to Section 4.)

Mizrahi and Moses [MM08] introduced continuous consensus and gave an efficient implementation called ConCon for the crash and sending omissions failure models. In addition to sending linear-sized messages per round, the ConCon protocol has a number of useful properties in the crash failure model:

- $M_i[k] = M_j[k]$ for all processes that are still active at time k, for all $k \geq 0$.

- There is a property W (standing for *waste*) of the particular failure pattern in a given execution, such that (i) $M_i[t+1-W]$ contains the input values of all active processes, and possibly of the failed processes, and (ii) $M_i[k]=\emptyset$ for all $k<t+1-W$.

- ConCon maintains the maximal core of all possible CC protocols: For every behavior of the adversary (determining the vector of inputs and every failure pattern), the core $M_i[k]$ at time k in ConCon is a superset of the core at time k in any other CC protocol, under the same adversary.

3 Fast Protocols

In this section, we show that ConCon yields fast protocols for simultaneous decision tasks.

There is a simple construction that transforms a decision protocol into a simultaneous decision protocol for the same problem with the same worst-case execution time. Let P be any decision task, and let A be any protocol that solves P. For any given time k, the protocol $\textsc{Delay}(A,k)$ is obtained from A simply by having processes delay decisions until time k. Thus, processes send exactly the same messages in A and $\textsc{Delay}(A,k)$. The only change is that whenever A specifies that a process should decide before time k, the process keeps track of its decision value and writes the value to its output registers only at time k. (Decisions after time k in A are performed unchanged.) Since processes communicate with each other via messages and not output registers, a process can change the time it writes to its output register without changing the views of other processes.

Lemma 1 *If protocol A solves decision task P, and $\hat{k}$ is an upper bound on the worst-case execution time of A, then* $\textsc{Delay}(A,\hat{k})$ *solves* $\textsc{sim}(P)$ *with execution time exactly $\hat{k}$.*

Proof: Completeness is satisfied since the nonfaulty processes will survive until time $\hat{k}$ and write to their output registers. Correctness is satisfied since the values chosen in a run of $\textsc{Delay}(A,\hat{k})$ are a subset of those chosen in A: every run r' of $\textsc{Delay}(A,\hat{k})$ maps to a run r of A where processes do not delay their decisions as they do in r', so there is a pair $(\vec{\imath},\vec{o})\in P$ such that $\vec{\imath}$ is the input vector for r (and hence r') and such that every process p_i that decides in r (and hence every process p_i that decides in r') chooses the value o_i in $\vec{o}$. Simultaneity is satisfied since all processes decide at time $\hat{k}$. ∎

There is a simple construction based on ConCon that transforms any decision protocol A into a simultaneous decision protocol ConCon(A) for the same problem with execution time $t+1-W$. Let P be a decision task, and let A be a protocol that solves P. In the protocol ConCon(A), each process $p_j \in \mathcal{P}$ follows the protocol ConCon until the core becomes nonempty. It then simulates an execution of A in which all processes whose initial values are in the core starts with these values and are fault-free, while all remaining processes are crashed and silent from the outset. The process then decides in ConCon(A) on the value it should decide on in the simulated run.

Theorem 2 *If protocol A solves the decision task P, then* ConCon(A) *solves* sim(P)*. In a run with failure pattern F, it decides at time $t+1-W(F)$.*

Proof: It follows from the discussion in Section 2.3 that ConCon satisfies several important properties: the core becomes nonempty for the first time at time $t+1-W$, all processes surviving to the end of round $t+1-W$ compute the same core at time $t+1-W$, and the initial states of all nonfaulty processes are in the core. Notice that the nonfaulty processes are nonfaulty in the simulated run of A, since the initial states of all nonfaulty processes are in the core, and hence at most t processes fail in the run of A and the remaining processes decide in this run of A. Completeness is satisfied since all nonfaulty processes are nonfaulty in the simulated run of A and decide. Simultaneity is satisfied since all processes learn that the core is nonempty for the first time at time $t+1-W$. Correctness is satisfied since all processes learn the same core at time $t+1-W$, and hence simulate the same run of A, and hence choose output values consistent with the input values according to the problem specification P. ∎

Given two solutions for a simultaneous decision task sim(P) that run in time $\hat{k}$ and $t+1-W$, respectively, we can compose them and run them in parallel to get a solution that decides simultaneously at the time which is the minimum of $\hat{k}$ and $t+1-W$. Moses and Raynal [MR08] define parallel composition as follows. Suppose that A and B are two protocols for a simultaneous decision task sim(P). Define A else B to be the protocol that runs A and B in parallel, but gives preference to A over B when choosing output values: In the composed protocol, process $p \in \mathcal{P}$ executes both protocols in parallel until the first round k at the end of which one of A and B has p decide. At that point, if only one protocol has p decide, then p

decides on the value determined by that protocol; and if both protocols have p decide, then p decides on the value determined by A.

Theorem 3 *Let $\hat{k}$ be an upper bound on the worst-case running time of a protocol A. If A solves P, then*

$$\textsc{Delay}(A, \hat{k}) \ \mathsf{else}\ \textsc{ConCon}(A)$$

solves $\textsc{sim}(P)$ *with execution time* $\min\{\hat{k}, t+1-W\}$.

Proof: First we prove that if A and B are simultaneous decision protocols, then in any run of A else B either all deciding processes choose according to A or all choose according to B. Suppose that p decides at time k_p and q decides at time k_q. Further assume (without loss of generality) that $k_p \leq k_q$. Whichever simultaneous protocol (A or B) caused p to decide at time k_p would also have caused q to decide at time k_p. So $k_q \leq k_p$ and it follows that $k_p = k_q$. Suppose that (without loss of generality) p decides at time k, and does so according to $X \in \{A, B\}$. If $X = A$ then A has q decide at time k as well, and so p and q both decide according to A in the composition. If $X = B$ then p does *not* decide before round k in either protocol, and does not decide according to A at time k. By the simultaneity of A and B, the same is true for q. Thus, q decides on the value determined by B in the composed protocol. It follows that in every run of $\textsc{Delay}(A, \hat{k})$ else $\textsc{ConCon}(A)$, either all deciding processes decide according to $\textsc{Delay}(A, \hat{k})$ or all decide according to $\textsc{ConCon}(A)$. Thus, (i) every run of the composition satisfies completeness, correctness, and simultaneity, and (ii) every run of the composition has processes decide at the earlier of $\hat{k}$ and $t + 1 - W$. ∎

4 Optimal Protocols

In this section, we prove that the protocol

$$\mathsf{Opt}(A, \hat{k}) \quad = \quad \textsc{Delay}(A, \hat{k}) \ \mathsf{else}\ \textsc{ConCon}(A)$$

for a decision task $\textsc{sim}(P)$ is not only fast, it is all-case optimal when $\hat{k}$ is the worst-case lower bound for P and A is a protocol that solves P in $\hat{k}$ rounds. To say that $\mathsf{Opt}(A, \hat{k})$ is *all-case optimal* means that for any input vector and failure pattern, $\mathsf{Opt}(A, \hat{k})$ decides as soon as any other protocol for $\textsc{sim}(P)$ would decide with the same input and failure pattern.

This generalizes a result by Moses and Raynal [MR08], shown for the particular decision task of condition-based consensus, where the condition (the set of possible input vectors) is assumed to satisfy a property called d-tightness. As in their case, we will use knowledge theory and known results about the structure of common knowledge to prove our claim. However, while proving optimality in [MR08] required a tailor-made lower bound argument, we present a novel proof technique that allows proving the claim at once for *all* decision problems. Before presenting the proof, we review just enough material from knowledge theory to support our proof.

Our lower bound is based on a well-known connection between simultaneous actions and common knowledge [DM90, MT88, MM08]. Rather than develop the logic of knowledge in detail here, we will employ a simple graph-theoretic interpretation of common knowledge that applies in our setting. For the rest of this section, fix a set $\mathcal{I}$ of input vectors, a number n of processes, and a bound $t \leq n-1$ on the number of failures. Define the *runs of A* to be the set of all runs of the form $A[\vec{\imath}, F]$ for all input vectors $\vec{\imath} \in \mathcal{I}$ and all failure patterns $F \in \mathsf{Fails}(t)$.

Similarity graph Given a protocol A, we say that two runs r and r' of A are *indistinguishable* to a process p at time k if process p survives round k in both runs and has the same local state at the end of round k in both runs. We define the *similarity graph* for A at time k to be the undirected graph where the vertices are the runs of A and the edges are all pairs $\{r, r'\}$ such that r and r' are indistinguishable to some process p at time k. We say that two runs r and r' of A are *connected* at time k if they are in the same connected component of the similarity graph for A at time k, which we denote by $r \overset{k}{\sim} r'$.

Common knowledge One way to define common knowledge is in terms of the connected components of the similarity graph [DM90]. Given a protocol A, a fact φ is *common knowledge* at time k in a run r of A if φ holds at time k in all runs r' of A satisfying $r' \overset{k}{\sim} r$. One can prove that if A solves $\textsc{sim}(P)$ and if processes decide at time k in a run r of A, then it is common knowledge at time k in r that processes are deciding at time k. Formulating this observation in terms of similarity, we have

Lemma 4 *Let P be a decision task and A be a protocol that solves P simultaneously. If the nonfaulty processes decide at time k in a run r of A, then they decide at time k in every run r' of A satisfying $r' \overset{k}{\sim} r$.*

Proof: It is enough to prove the result for the case of a single edge from r to r' in the similarity graph at time k, and the result will follow by induction since $r \overset{k}{\sim} r'$ means there is a finite path of edges from r to r'. Since there is an edge from r to r', there is a process p that survives round k in both runs and has the same local state at the end of round k in both runs. Since the nonfaulty processes decide at time k in r and p has not failed, process p must decide at time k in r. Since p has the same local state at time k in both runs, it must decide at time k in r' as well. Since decisions are simultaneous in runs of A, all nonfaulty processes must decide at time k in r', and we are done. ∎

Waste It is known that an adversarial scheduler can keep a fact from becoming common knowledge by failing processes that know this fact, and the best strategy for the adversary is to fail one process per round to keep a fact from becoming common knowledge until the end of round $t+1$. To fail more than one process per round is a waste. Following [DM90], we capture this intuition as follows. Given a failure pattern F, we say that the failure of a process p is *exposed* in round k if the round k edge from p to q is missing in F for some process q that survives round k in F. Let $E(F,k)$ be the number of processes whose failure is exposed in round k or earlier. Observe that $E(F,0)=0$ for all $F \in \mathsf{Fails}(t)$. Let $W(F)$ denote the *waste* inherent in F defined by

$$W(F) = \max_{k\geq 0}\{E(F,k)-k\}.$$

Notice that $0 \leq W(F) \leq t-1$ for all $F \in \mathsf{Fails}(t)$. In the language of [MR09], we say that round k is *premature* in a run $r = A[\vec{\imath},F]$ if $k < t+1-W(F)$, since we shall see that no nontrivial fact can be be common knowledge at the end of a premature round. Specifically, the analysis of connectivity in the similarity graph performed in [DM90] showed the following. We say that the set $\mathcal{I}$ of input vectors is *complete* if it is equal to a Cartesian product $I_1 \times \cdots \times I_n$, meaning that process input values can be chosen independently.

Lemma 5 [DM90, MR08] *Suppose the set $\mathcal{I}$ of input vectors is complete. If round k is premature in two runs r and r' of A, then $r \overset{k}{\sim} r'$.*

We now have the combinatorial machinery that we need to prove our main result:

Theorem 6 (All-case optimality) *Let P be a decision task with worst-case lower bound $\hat{k}$ and let A be a protocol that solves P in time $\hat{k}$. If P's* *set of input vectors is complete, then*

$$\mathsf{Opt}(A,\hat{k}) = \textsc{Delay}(A,\hat{k}) \ \mathsf{else}\ \textsc{ConCon}(A)$$

is a protocol for $\textsc{sim}(P)$ *that is all-case optimal.*

Proof: Since A solves P in time $\hat{k}$, Theorem 3 implies that $\mathsf{Opt}(A,\hat{k})$ solves $\textsc{sim}(P)$ in time $\min\{\hat{k}, t+1-W(F)\}$, where F is the failure pattern.

Suppose that the theorem is false: that $\mathsf{Opt}(A,\hat{k})$ is not all-case optimal. This means there is a protocol B solving $\textsc{sim}(P)$ and an input vector $\vec{\imath} \in \mathcal{I}$ and a failure pattern $F \in \mathsf{Fails}(t)$ such that processes decide at time $k_0 < \min\{\hat{k}, t+1-W(F)\}$ in the run $r = B[\vec{\imath}, F]$. Let $\hat{B} = \mathsf{Opt}(B, k_0)$. Notice that in the run $\hat{r} = \hat{B}[\vec{\imath}, F]$ corresponding to r, processes decide according to B at time k_0 since $k_0 < t+1-W(F)$ is premature in $\hat{r}$. We now prove that $\hat{B}$ solves $\textsc{sim}(P)$ within at most $k_0 < \hat{k}$ rounds, contradicting the assumption that $\hat{k}$ is a worst-case lower bound for P.

Since B solves $\textsc{sim}(P)$ by assumption and $\textsc{ConCon}(B)$ solves $\textsc{sim}(P)$ by Theorem 2, any decision by $\textsc{Delay}(B, k_0)$ or $\textsc{ConCon}(B)$ is correct and simultaneous, and hence any decision by $\hat{B}$ is correct and simultaneous. We need only prove that one of $\textsc{Delay}(B, k_0)$ or $\textsc{ConCon}(B)$ actually makes a decision at or before time k_0 in every run, and hence that $\hat{B}$ does so as well. Let $\hat{r}' = \hat{B}[\vec{\jmath}, F']$ be any run of $\hat{B}$. Let $k_1 = t+1-W(F')$ and consider two cases:

- Suppose that $k_1 \le k_0$. Theorem 2 says $\textsc{ConCon}(B)$ decides at time $k_1 = t+1-W(F')$ in $\hat{r}' = \hat{B}[\vec{\jmath}, F']$, so $\hat{B}$ can decide at time $k_1 \le k_0$ in $\hat{r}'$.

- Suppose that $k_1 > k_0$. In this case, k_0 is premature in both $\hat{r}$ and $\hat{r}'$, and thus $\hat{r} \overset{k_0}{\sim} \hat{r}'$ by Lemma 5. Since processes decide at k_0 in $\hat{r}$, they decide at k_0 in $\hat{r}'$ by Lemma 4. ∎

The proof shows that beating the worst-case lower bound is as hard as solving consensus. In particular, if $\textsc{Delay}(A,\hat{k})$ solves $\textsc{sim}(P)$ at time $k < \hat{k}$, then $k \ge t+1-W$ at which time consensus can be solved.

Coverability Theorem 6 requires that the set of input vectors is complete, meaning that is a Cartesian product, which is typically true of most decision tasks. We can generalize this theorem using an inherently topological notion of coverability. A set $\mathcal{I}$ of input vectors is *c-coverable* if for every pair of input vectors $\vec{\imath}$ and $\vec{\jmath}$ in $\mathcal{I}$ there is a finite sequence of input vectors $\vec{\imath} = \vec{\imath}_0, \vec{\imath}_1, \ldots, \vec{\imath}_h = \vec{\jmath}$ in $\mathcal{I}$ with the property that adjacent vectors $\vec{\imath}_\ell$ and $\vec{\imath}_{\ell+1}$ differ on the inputs of at most c processes. Note that if $\mathcal{I}$ is complete (a Cartesian product), then $\mathcal{I}$ is 1-coverable. We can generalize Lemma 5 from complete to c-coverable sets of input vectors:

Lemma 7 [MR08] *Suppose the set $\mathcal{I}$ of input vectors is c-coverable. If $k \leq t+1-c$ and round k is premature in two runs r and r' of A, then $r \overset{k}{\sim} r'$.*

We can generalize Theorem 6 from complete to c-coverable sets of input vectors:

Theorem 8 (All-case optimality) *Let P be a decision task with worst-case lower bound $\hat{k}$ and let A be a protocol that solves P in time $\hat{k}$. If P's set of input vectors is c-coverable and $\hat{k} \leq t + 2 - c$, then*

$$\mathsf{Opt}(A, \hat{k}) = \textsc{Delay}(A, \hat{k}) \text{ else } \textsc{ConCon}(A)$$

is a protocol for $\textsc{sim}(P)$ that is all-case optimal.

Proof: Simply use Lemma 7 in place of Lemma 5 in the proof of Theorem 6. The only tricky observation is that Lemma 5 is applied with $k = k_0 < \hat{k} \leq t + 2 - c$ which implies the hypothesis $k \leq t + 1 - c$ required by the lemma. ∎

5 Applications

The construction of Theorem 6 yields simultaneous protocols that are all-case optimal for some of the most famous problems in distributed computation.

5.1 Set agreement and consensus

The k-set agreement problem [Cha90] is a well-known generalization of consensus [PSL80, LSP82]. Given a set V of at least $k + 1$ values, processes start with inputs from V and must choose outputs from V subject to three requirements:

- *Termination*: Every nonfaulty process chooses an output value.
- *Validity*: Every process's output value is some process's input value.

- *Agreement*: The set of output values chosen must contain at most k distinct values.

The sets $\mathcal{I}$ and $\mathcal{O}$ of input and output vectors are $V \times \cdots \times V$ (n copies of V). Consensus is k-set agreement with $k = 1$.

Set agreement is most famous for a trio of papers [HS99, BG93, SZ00] proving that set agreement is impossible in asynchronous systems, generalizing the impossibility result for consensus [FLP85]. One paper [CHLT00], however, proves that $\lfloor t/k \rfloor + 1$ is a tight worst-case bound on the number of rounds required for k-set agreement in the synchronous model. This matches and generalizes the tight bound of $t+1$ rounds for consensus [FL81, DS82]. Let SA be a k-set agreement protocol that halts in $\lfloor t/k \rfloor + 1$ rounds. Theorem 6 implies:

Corollary 9 $\mathsf{Opt}(\textsc{sa}, \lfloor t/k \rfloor + 1)$ *is a protocol for k-set agreement that is all-case optimal. It halts in time*

$$\min\{\lfloor t/k \rfloor + 1, t + 1 - W\}.$$

5.2 Condition-based consensus

Consensus and k-set agreement are decision tasks whose set $\mathcal{I} = V \times \cdots \times V$ of input vectors allows any process to start with any value in V. Condition-based consensus was defined [MRR03] as a way of circumventing the impossibility of consensus in asynchronous models [FLP85] by restricting the set $\mathcal{I}$ of input vectors to a subset of $V \times \cdots \times V$. The intuition is that consensus is easier to solve when fewer input vectors are possible. A protocol solves condition-based consensus for a given condition $\mathcal{I}$ if all of its executions satisfy the termination, agreement, and validity conditions for 1-set agreement (consensus).

Every subset $\mathcal{I}$ of $V \times \cdots \times V$ defines a condition, and hence defines an instance of condition-base consensus. A property of conditions called *d-legality* was defined in [MRR06], and a protocol was presented that solves condition-based consensus for all d-legal conditions in $t + 1 - d$ rounds. However, not all d-legal conditions require $t + 1 - d$ rounds in the worst case. A stronger property called *d-tightness* was defined in [MR08] to mean both d-legal and $(d+1)$-coverable. An example of a d-tight condition is the set M_t^d of all vectors such that the largest value appearing in the vector appears there more than d times.

The results of [MR08] imply that for every d-tight condition $\mathcal{I}_d$, there is a worst-case lower bound of $t + 1 - d$ rounds for decision in condition-based consensus. The results of [MRR06] imply there is a condition-based consensus protocol that halts in $t + 1 - d$ rounds on d-legal (and hence on d-tight) conditions. Let $\textsc{cbc}(d)$ be any such condition-based consensus protocol. Theorem 8 implies:

Corollary 10 $\mathsf{Opt}(\textsc{cbc}(d), t + 1 - d)$ *is a protocol for condition-based consensus that is all-case optimal on d-tight conditions. It halts in time*

$$\min\{t + 1 - d, t + 1 - W\}.$$

Proof: Apply Theorem 8 with $c = d + 1$ and $\hat{k} = t + 1 - d$. The input vectors are c-coverable because they are d-tight and thus $(d + 1)$-coverable. The bound $\hat{k} \leq t + 2 - c$ holds because $\hat{k} = t + 1 - d$. ∎

We note that Corollary 10 is the central result of Moses and Raynal in [MR08]. While their proof required a careful and nontrivial explicit lower bound proof for condition-based consensus, ours is obtained in a more uniform manner.

5.3 Renaming

The renaming and strong renaming problems were first defined in the asynchronous model [ABND$^+$90]. In the renaming problem, processes start with distinct names from a large namespace and are required to choose distinct names from a small namespace, a namespace of size roughly equal to the number of processes participating in the protocol. In the strong renaming problem, processes are required to preserve the order of names: if p and q start with names $i_p < i_q$, then they are required to choose names $o_p < o_q$. In both problems, given sets I and O of initial and final names, the set $\mathcal{I}$ is the subset of $I \times \cdots \times I$ consisting of vectors of distinct names, and the set $\mathcal{O}$ is the set $O \times \cdots \times O$.

The first paper to consider strong renaming in the synchronous model was [HT90]. They proved a tight $\log c$ worst-case bound for strong renaming, where c is the number of processes concurrently participating in the protocol. Since c depends on the execution, their results imply a worst-case bound of $\log n$. Let $\textsc{sr}$ be a strong renaming protocol that halts in $\log n$ rounds. Since the input vectors are 1-coverable, Theorem 8 implies:

Corollary 11 $\mathsf{Opt}(\textsc{sr}, \log n)$ *is a protocol for strong renaming that is all-case optimal assuming* $\log n \leq t + 1$. *It halts in time*

$$\min\{\log n, t + 1 - W\}.$$

Proof: Apply Theorem 8 with $c = 1$ and $\hat{k} = \log n$. The input vectors are 1-coverable. The bound $\hat{k} \leq t + 2 - c$ holds because $\hat{k} = \log n \leq t + 1$. ∎

A few comments on $\mathsf{Opt}(\textsc{sr}, \log n)$ are in order. While the original protocol SR of [HT90] allows a subset of the nonfaulty processes not to participate in the execution, the optimal protocol $\mathsf{Opt}(\textsc{sr}, \log n)$ does not. This may be justified by the intuition that a protocol that must be optimally fast under all conditions cannot be expected to allow some of the processes be dormant, unless their identity is built into the protocol.

6 Conclusions

This work shows how knowledge and topology can be used together to attain interesting results about distributed computation, and begins what we hope will be a fruitful approach to reasoning about distributed computation. While the proofs we presented in this paper are combinatorial, the definitions and results used in the paper all come from or are inspired by knowledge and topology.

Our technical results concern the construction of simultaneous protocols for decision tasks. We have demonstrated that the protocol ConCon derived by knowledge-theoretic means can be used to solve any decision task simultaneously, and running it in parallel with a protocol A solving the decision task can solve the task faster than A or ConCon alone, and that if A matches the worst-case bound, the parallel composition yields an all-case optimal solution to the task.

Our primary insights, however, come from the proofs of the technical results:

- *Stopping early requires attaining common knowledge of a nontrivial fact.* Every protocol has a worst-case run $\hat{r}$ in which processes decide at a time $\hat{k}$ late in the run. Consider any other run r in which processes decide earlier time k. Lemma 4 says that processes decide at time k in every run r' connected to r. It follows that $\hat{r}$ cannot be connected to r since processes do not decide at time $k < \hat{k}$ in $\hat{r}$. So it is common knowledge at time k in the run r that the current run is not connected to $\hat{r}$, a nontrivial fact.

- *Stopping early means consensus can be solved.* Stopping early implies a nontrivial fact has become common knowledge. The ConCon protocol computes a core that characterizes all nontrivial facts that are common knowledge. Stopping early means the core has become nonempty, and it is known that consensus can be solve once the core becomes nonempty.

Notice that the conclusion of the first point—common knowledge that the current run is not connected to the worst-case run—is inherently a statement about knowledge of topology (connectivity). We hope the combination of knowledge and topology will yield many more insights in the years to come.

Acknowledgments

We thank our anonymous referees for their helpful comments and advice.

References

[ABND+90] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk. Renaming in an asynchronous environment. *Journal of the ACM*, 37(3):524–548, 1990.

[AR02] H. Attiya and S. Rajsbaum. The combinatorial structure of wait-free solvable tasks. *SIAM Journal on Computing*, 31(4):1286–1313, 2002.

[BG93] E. Borowsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 91–100, 1993.

[Cha90] S. Chaudhuri. Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 311–324, 1990.

[CHLT00] S. Chaudhuri, M. Herlihy, N. A. Lynch, and M. R. Tuttle. Tight bounds for k-set agreement. *Journal of the ACM*, 47(5):912–943, 2000.

[DM90] C. Dwork and Y. Moses. Knowledge and common knowledge in a Byzantine environment: Crash failures. *Information and Computation*, 88(2):156–186, 1990.

[DS82] D. Dolev and H. R. Strong. Polynomial algorithms for multiple processor agreement. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 401–407, 1982.

[FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, MA, 1995.

[FL81] M. J. Fischer and N. A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14:183–186, 1981.

[FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM*, 32(2):374–382, 1985.

[HM90] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.

[HS99] M. Herlihy and N. Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858–923, 1999.

[HT90] M. Herlihy and M. R. Tuttle. Wait-free computation in message-passing systems. In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 347–362, August 1990.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. on Programming Languages and Systems*, 4(3):382–401, 1982.

[MM08] T. Mizrahi and Y. Moses. Continuous consensus via common knowledge. *Distributed Computing*, 20(5):305–321, 2008.

[MR08] Y. Moses and M. Raynal. No double discount: Condition-based simultaneity yields limited gain. In *Proc. 22nd Int. Symp. on Distributed Computing*, pages 423–437, September 2008.

[MR09] Y. Moses and M. Raynal. Revisiting simultaneous consensus with crash failures. *Journal of Parallel and Distributed Computing*, 69(4):400–409, 2009.

[MRR03] A. Mostefaoui, S. Rajsbaum, and M. Raynal. Conditions on input vectors for consensus solvability in asynchronous distributed systems. *Journal of the ACM*, 50(6):922–954, 2003.

[MRR06] A. Mostéfaoui, S. Rajsbaum, and M. Raynal. Synchronous condition-based consensus. *Distributed Computing*, 18(5):325–343, April 2006.

[MT88] Y. Moses and M. R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3:121–169, 1988.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[SZ00] M. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM Journal on Computing*, 29(5):1449–1483, 2000.