A Tight Lower Bound for k-Set Agreement

Soma Chaudhuri* Maurice Herlihy† Nancy A. Lynch‡ Mark R. Tuttle§

Abstract: We prove tight bounds on the time needed to solve *k-set agreement*, a natural generalization of *consensus*. We analyze this problem in a synchronous, message-passing model where processors fail by crashing. We prove a lower bound of $\lfloor f/k \rfloor + 1$ rounds of communication for solutions to k-set agreement that tolerate f failures. This bound is tight, and shows that there is an inherent tradeoff between the running time, the degree of coordination required, and the number of faults tolerated, even in idealized models like the synchronous model. The proof of this result is interesting because it is a geometric combination of other well-known proof techniques.

1 Introduction

Most interesting problems in concurrent and distributed computing require processors to coordinate their actions in some way. It can also be important for protocols solving these problems to tolerate processor failures, and to execute quickly. Ideally, one would like to optimize all three properties—degree of coordination, fault-tolerance, and efficiency—but in practice, of course, it is usually necessary to make tradeoffs among them. In this paper, we give a precise characterization of the tradeoffs required by studying a family of basic coordination problems called k-set agreement.

In *k-set agreement* [Cha91], each processor starts with an arbitrary input value and halts after choosing an output value. These output values must satisfy two conditions: each output value must be some processor's input value, and the set of output values chosen must contain at most k distinct values. The first condition rules out trivial solutions in which a single value is hard-wired into the protocol and chosen by all processors in all executions, and the second condition requires that the processors coordinate their choices to some degree. This problem is interesting because it defines a family of coordination problems of increasing difficulty. At one extreme, if n is the number of processors in the system, then n-set agreement is trivial: each processor may simply choose its own input value. At the other extreme, 1-set agreement requires that all processors choose the same output value, a problem equivalent to the *consensus* problem [LSP82, PSL80, FL82, FLP85, Dol82, Fis83]. Consensus is well-known to be the "hardest" problem, in the sense that all other decision problems can be reduced to it.[1] Between these extremes, as we vary the value of k from n to 1, we gradually increase the degree of processor coordination required.

We consider this family of problems in a *synchronous*, *message-passing* model with *crash failures*. In this model, n processors communicate by sending messages over a completely connected network. Computation in this model proceeds in a sequence of rounds. In each round, processors send messages to other processors, then receive messages sent to them in the same round, and then perform some local computation and change state. This means that all processors take steps at the same rate, and that all messages take the same amount of time to be delivered. Communication is reliable, but up to f processors can fail by stopping in the middle of the protocol.

In this model, we prove that any protocol solving k-set agreement and tolerating f failures requires $\lfloor f/k \rfloor + 1$ rounds of communication in the worst case,

*226 Atanasoff Hall, Dept of Computer Science, Iowa State University, Ames, IA 50011; chaudhur@cs.iastate.edu.

†DEC Cambridge Research Lab, One Kendall Sq, Bldg 700, Cambridge, MA 02139; herlihy@crl.dec.com.

‡MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139; lynch@theory.lcs.mit.edu.

§DEC Cambridge Research Lab, One Kendall Sq, Bldg 700, Cambridge, MA 02139; tuttle@crl.dec.com.

This paper will appear in *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, Palo Alto, CA, November, 1993.

[1] Consensus arises in applications as diverse as on-board aircraft control [W+78], database transaction commit [BHG87], and concurrent object design [Her91].

assuming $n \geq f + k + 1$. This lower bound is tight, matching a protocol given by Chaudhuri [Cha91]. Since consensus is just 1-set agreement, our lower bound implies the well-known lower bound of $f + 1$ rounds for consensus when $n \geq f + 2$ [FL82]. More important, the running time $r = \lfloor f/k \rfloor + 1$ demonstrates that there is a smooth but inescapable tradeoff among the number f of faults tolerated, the degree k of coordination achieved, and the time r the protocol must run. In addition, the lower bound proof itself is interesting because of the geometric proof technique we use, combining ideas due to Chaudhuri [Cha91, Cha93], Fischer and Lynch [FL82], Herlihy and Shavit [HS93], and Dwork, Moses, and Tuttle [DM90, MT88].

The synchronous model is a special case of almost every other realistic model, so any lower bound in this model holds in these models as well. Moreover, our techniques may be helpful in understanding how to prove (possibly) stricter lower bounds in these more complex models.

2 Overview

We start with an informal overview of the ideas used in the lower bound proof. For the remainder of this paper, suppose P is a protocol that solves k-set agreement and tolerates the failure of f out of n processors, and suppose P halts in $r < \lfloor f/k \rfloor + 1$ rounds. This means that all nonfaulty processors have chosen an output value at time r in every execution of P. In addition, suppose $n \geq f + k + 1$, which means that at least $k + 1$ processors never fail. Our goal is to consider the *global states* that occur at time r in executions of P, and to show that in one of these states there are $k + 1$ processors that have chosen $k + 1$ distinct values, violating k-set agreement. Our strategy is to consider the *local states* of processors that occur at time r in executions of P, and to investigate the combinations of these local states that occur in global states. This investigation depends on the construction of a geometric object. In this section, we use a simplified version of this object to illustrate the general ideas in our proof.

Since consensus is a special case of k-set agreement, it is helpful to review the standard proof of the $f + 1$ round lower bound for consensus [FL82, DS83, Mer85, DM90] to see why new ideas are needed for k-set agreement. Suppose that the protocol P is a consensus protocol, which means that in all executions of P all nonfaulty processors have chosen the same output value at time r. Two global states g_1 and g_2 at time r are said to be *similar* if some nonfaulty processor p has the same local state in both global states. The crucial property of similarity is that the decision value of any processor in one global state completely determines the decision value for any processor in all similar global states. For example, if all processors decide v in g_1, then certainly p decides v in g_1. Since p has the same local state in g_1 and g_2, and since p's decision value is a function of its local state, processor p also decides v in g_2. Since all processors agree with p in g_2, all processors decide v in g_2, and it follows that the decision value in g_1 determines the decision value in g_2. A *similarity chain* is a sequence of global states, $g_1, \ldots, g_\ell$, such that g_i is similar to g_{i+1}. A simple inductive argument shows that the decision value in g_1 determines the decision value in g_ℓ. The lower bound proof consists of showing that all time r global states of P lie on a single similarity chain. It follows that all processors choose the same value in all executions of P, independent of the input values, violating the definition of consensus.

The problem with k-set agreement is that the decision values in one global state do not determine the decision values in similar global states. If p has the same local state in g_1 and g_2, then p must choose the same value in both states, but the values chosen by the other processors are not determined. Even if $n - 1$ processors have the same local state in g_1 and g_2, the decision value of the last processor is still not determined. The fundamental insight in this paper is that k-set agreement requires considering all "degrees" of similarity at once. We capture these degrees of similarity with a compact *geometric* generalization of similarity chains.

We start with a k-dimensional *simplex* in k-dimensional Euclidean space [Cha93, HS93]. A simplex is just the natural generalization of a triangle to k dimensions: for example, a 0-dimensional simplex is a vertex, a 1-dimensional simplex is an edge linking two vertices, a 2-dimensional simplex is a solid triangle, and a 3-dimensional simplex is a solid tetrahedron. The simplex contains a number of *grid points*, which are the points in Euclidean space with integer coordinates. We *triangulate* this simplex with respect to these grid points via a collection of smaller k-dimensional simplexes. We call this triangulated simplex the *Bermuda Triangle* B, since all fast protocols vanish somewhere in its interior. We then label each grid point with an ordered pair (p, s) consisting of a processor identifier p and a local state s in such a way that for each simplex T in the triangulation there is a global state g consistent with the labeling of the simplex: for each ordered pair (p, s) labeling a corner of T, processor p has local state s in global state g.

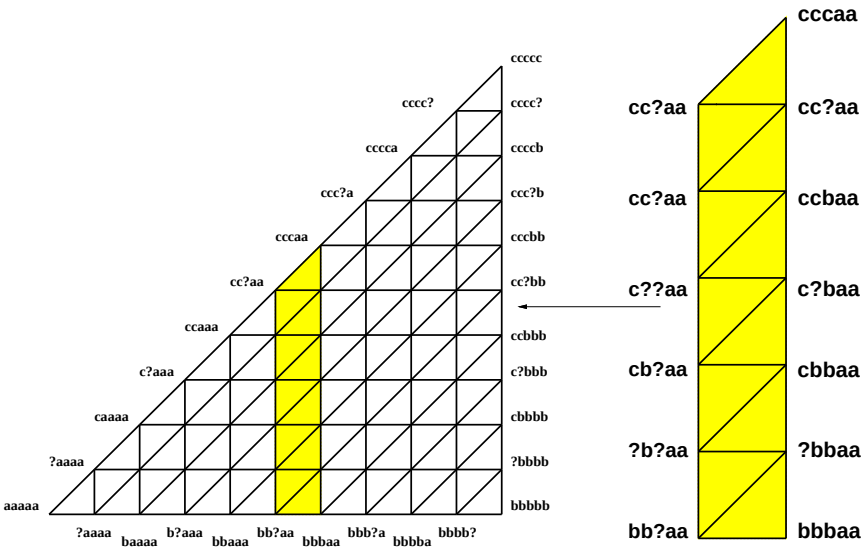

Figure 1: The Bermuda Triangle for 5 processors and a 1-round protocol for 2-set agreement.

A simplified Bermuda Triangle B is illustrated in Figure 1. In this figure, P is a protocol for 5 processors solving 2-set agreement in 1 round. We have labeled grid points with local states, but we have omitted processor ids and many intermediate nodes for clarity. The local states in the figure are represented by expressions such as $bb?aa$. Given 3 distinct input values a, b, c, we write $bb?aa$ to denote the local state of a processor p at the end of a round in which the first two processors have input value b and send messages to p, the middle processor fails to send a message to p, and the last two processors have input value a and send messages to p. In Figure 1, following any horizontal line from left to right across B, the input values are changed from a to b. The input value of each processor is changed—one after another—by first silencing the processor, and then reviving the processor with the input value b. Similarly, moving along any vertical line from bottom to top, processors' input values change from b to c.

The complete labeling of the Bermuda Triangle B—which would include processor ids—has the following property. Let (p, s) be the label of a grid point x. If x is a corner of B, then s specifies that each processor starts with the same input value, so p must choose this value if it finishes protocol P in local state s. If x is on an edge of B, then s specifies that each processor starts with one of the two input values labeling the ends of the edge, so p must choose one of these values if it halts in state s. Similarly, if x is in the interior of B, then s specifies that each processor starts with one of the three values labeling the corners of B, so p must choose one of these three values if it halts in state s.

Now let us "color" each grid point with output values. Given a grid point x labeled with (p, s), let us color x with the value v that p chooses in local state s at the end of P. This coloring of B has the property that the color of each of the corners is determined uniquely, the color of each point on an edge between two corners is forced to be the color of one of the corners, and the color of each interior point can be the color of any corner. Colorings with this property are called *Sperner colorings*, and have been studied extensively in the field of algebraic topology. At this point, we exploit a remarkable combinatorial result first proved in 1928: *Sperner's Lemma* [Spa66, p.151] states that any Sperner coloring of any triangulated k-dimensional simplex must include at least one simplex whose corners are colored with all $k + 1$ colors. In our case, however, this simplex corresponds to a global state in which $k+1$ processors choose $k+1$ distinct values, which contradicts the definition of k-set agreement. Thus, in the case illustrated above, there is no protocol for 2-set agreement halting in 1 round.

We note that the idea of applying Sperner's Lemma to a geometric structure like the Bermuda Triangle has been used in previous work by Chaudhuri [Cha91, Cha93]. She also proves a lower bound of $\lfloor f/k \rfloor + 1$ rounds for k-set agreement, but for a very restricted class of protocols. In her work, a protocol's decision function can depend only on vectors giving partial information about which processors started with which inputs, but cannot make any use of any other

information recorded in a processor's local state. The technical challenge in this paper is to construct a labeling of vertices with processor ids and local states that will allow us to prove a lower bound for arbitrary protocols, including protocols that have processors make arbitrary use of the information in their local states.

3 The Model

We use a synchronous, message-passing model with crash failures. The system consists of n processors, $p_1, \ldots, p_n$. Processors share a global clock that starts at 0 and advances in increments of 1. Computation proceeds in a sequence of *rounds*, with round r lasting from time $r-1$ to time r. Computation in a round consists of three phases: first each processor p sends messages to some of the processors in the system, possibly including itself, then it receives the messages sent to it during the round, and finally it performs some local computation and changes state. We assume that the communication network is totally connected: every processor is able to send distinct messages to every other processor in every round. We also assume that communication is reliable (although processors can fail): if p sends a message to q in round r, then the message is delivered to q in round r.

Processors follow a deterministic *protocol* that determines what messages a processor should send and what output a processor should generate as a function of its local state. Processors can be faulty, however, and any processor p can simply *stop* in any round r. In this case, processor p follows its protocol and sends all messages the protocol requires in rounds 1 through $r-1$, sends some subset of the messages it is required to send in round r, and sends no messages in rounds after r. We say that p is *silent* from round r if p sends no messages in round r or later. We say that p is *active* through round r if p sends all messages in round r and earlier.

A *full-information protocol* is one in which every processor broadcasts its entire local state to every processor, including itself, in every round [PSL80, FL82, Had83]. One nice property of full-information protocols is that every execution of a full-information protocol P has a compact representation called a *communication graph* [MT88]. The communication graph $\mathcal{G}$ for an r-round execution of P is a two-colored graph. The vertices form an $n \times r$ grid, with processor names 1 through n labeling the vertical axis and times 0 through r labeling the horizontal axis. The node representing processor p at time i is labeled with the pair $\langle p, i \rangle$. Given any pair of processors p and q and any round i, there is an edge between $\langle p, i-1 \rangle$ and $\langle q, i \rangle$ whose color determines whether p successfully sends a message to q in round i: the edge is green if p succeeds, and red otherwise. In addition, each node $\langle p, 0 \rangle$ is labeled with p's input value. Figure 2 illustrates a three round communication graph; in this figure, only green edges are indicated explicitly. We refer to the edge between $\langle p, i-1 \rangle$ and $\langle q, i \rangle$ as the *round i edge from p to q*, and we refer to the node $\langle p, i-1 \rangle$ as the *round i node for p* since it represents to point at which p sends its round i messages. We define what it means for a processor to be *silent* or *active* in terms of communication graphs in the obvious way. In the stopping failure model, all communication graphs have the property that if a round i edge from p is red, then all round $j \geq i+1$ edges from p are red (meaning p is silent from round $i+1$).

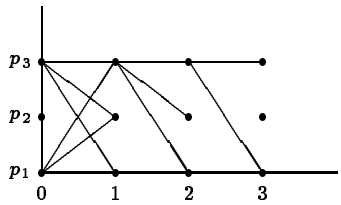

Figure 2: A three-round communication graph.

Since a communication graph $\mathcal{G}$ describes an execution of P, it also determines the global state at the end of P, so we sometimes refer to $\mathcal{G}$ as a *global communication graph*. In addition, for each processor p and time t, there is a subgraph of $\mathcal{G}$ that corresponds to the local state of p at the end round t, and we refer to this subgraph as a *local communication graph*. The local communication graph for p at time t is the subgraph $\mathcal{G}(p,t)$ of $\mathcal{G}$ induced by the node $\langle p, t \rangle$ and all earlier nodes reachable from $\langle p, t \rangle$ by a sequence (directed backwards in time) of green edges followed by at most one red edge. In the remainder of this paper, we use graphs to represent states. Wherever we used "state" in the informal overview of Section 2, we now substitute the word "graph." Furthermore, we assume that all executions of a full-information protocol run for exactly r rounds and produce output at exactly time r, and we assume that processors send local communication graphs instead of local states.

The crucial property of a full-information protocol is that every r-round protocol for k-set agreement can be simulated by an r-round full-information protocol, and hence that we can restrict attention to full-information protocols when proving the lower bound in this paper (see [PSL80, FL82, Had83, MT88]).

4 k-set Agreement

The *k-set agreement problem* [Cha91] is defined as follows. We assume that each processor p_i has two private registers in its local state, a read-only input register and a write-only output register. Initially, p_i's input register contains an arbitrary input value from a set V containing at least $k+1$ values $v_0, \ldots, v_k$, and its output register is empty. A protocol solves the problem if it causes each nonfaulty processor to halt after writing an output value to its output register in such a way that every processor's output value is some processor's input value, and the set of output values chosen has size at most k.

5 Bermuda Triangle

In this section, we define the basic geometric constructs used in our proof that every protocol P solving k-set agreement and tolerating f failures requires at least $\lfloor f/k \rfloor + 1$ rounds of communication, assuming $n \geq f + k + 1$.

We start with some preliminary definitions. A *simplex* S is the convex hull of $k+1$ affinely-independent[2] points $x_0, \ldots, x_k$ in Euclidean space. This simplex is a k-dimensional volume, the k-dimensional analogue of a solid triangle or tetrahedron. The points $x_0, \ldots, x_k$ are called the *vertices* of S, and k is the *dimension* of S. We sometimes call S a *k-simplex* when we wish to emphasize its dimension. A simplex F is a *face* of S if the vertices of F form a subset of the vertices of S (which means that the dimension of F is less than or equal to the dimension of S). A set of k-simplexes $S_1, \ldots, S_\ell$ is a *triangulation* of S if $S = S_1 \cup \cdots \cup S_\ell$ and the intersection of S_i and S_j is a face of each[3] for all pairs i and j. The *vertices* of a triangulation are the vertices of the S_i. Any triangulation of S induces triangulations of its faces in the obvious way.

Let $\mathcal{B}$ be the k-simplex in k-dimensional Euclidean space with vertices $(0, \ldots, 0)$, $(N, 0, \ldots, 0)$, $(N, N, 0, \ldots, 0)$, $\ldots$, $(N, \ldots, N)$, where N is a huge integer defined later in Section 6.3. The *Bermuda Triangle* B is a triangulation of $\mathcal{B}$ defined as follows. The vertices of B are the grid points contained in $\mathcal{B}$: these are the points of the form $x = (x_1, \ldots, x_k)$, where the x_i are integers between 0 and N satisfying $x_1 \geq x_2 \geq \cdots \geq x_k$. Informally, the simplexes of the triangulation are defined as follows: pick any grid point and walk one step in the positive direction along each dimension. The $k+1$ points visited by this walk define the vertices of a simplex, and the triangulation B consists of all simplexes determined by such walks. For example, the 2-dimensional Bermuda Triangle is illustrated in Figure 1. This triangulation, known as *Kuhn's triangulation*, is defined formally as follows [Cha93]. Let $e_1, \ldots, e_k$ be the unit vectors; that is, e_i is the vector $(0, \ldots, 1, \ldots, 0)$ with a single 1 in the ith coordinate. A simplex is determined by a point y_0 and an arbitrary permutation $f_1, \ldots, f_k$ of the unit vectors $e_1, \ldots, e_k$: the vertices of the simplex are the points $y_i = y_{i-1} + f_i$ for all $i > 0$. When we list the vertices of a simplex, we always write them in the order $y_0, \ldots, y_k$ in which they are visited by the walk.

For brevity, we refer to the vertices of $\mathcal{B}$ as the *corners* of B. The triangulation B induces triangulations of the one-dimensional faces of $\mathcal{B}$, and these induced triangulations are called the *edges* of B. The simplexes of B are called *primitive simplexes*.

Each vertex of B is labeled with an ordered pair $(p, \mathcal{L})$ consisting of a processor id p and a local communication graph $\mathcal{L}$. As illustrated in the overview in Section 2, the crucial property of this labeling is that if S is a primitive simplex with vertices $y_0, \ldots, y_k$, and if each vertex y_i is labeled with a pair $(q_i, \mathcal{L}_i)$, then there is a global communication graph $\mathcal{G}$ such that each q_i is nonfaulty in $\mathcal{G}$ and has local communication graph $\mathcal{L}_i$ in $\mathcal{G}$. Constructing this labeling is the subject of the next three sections. We first assign global communication graphs $\mathcal{G}$ to vertices in Section 6, then we assign processors p to vertices in Section 7, and then we assign ordered pairs $(p, \mathcal{L})$ to vertices in Section 8, where $\mathcal{L}$ is the local communication graph of p in $\mathcal{G}$.

6 Graph Assignment

In this section, we label each vertex of B with a global communication graph. Actually, for expository reasons, we augment the definition of a communication graph and label vertices of B with these augmented communication graphs instead. Constructing this labeling involves several steps.

6.1 Augmented Graphs

In this section, we extend the definition of a communication graph to make the processor assignment in Section 7 easier to describe. We augment communication graphs with tokens, and place tokens on the graph so that if processor p fails in round i, then there is a token on the node $\langle p, j-1 \rangle$ for processor

[2] Points $x_0, \ldots, x_k$ are affinely independent if $x_1 - x_0, \ldots, x_k - x_0$ are linearly independent.

[3] Notice that the intersection of two arbitrary k-dimensional simplexes S_i and S_j will be a volume of some dimension, but it need not be a face of either simplex.

p in some earlier round $j \leq i$. In this sense, every processor failure is "covered" by a token, and the number of processors failing in the graph is bounded from above by the number of tokens. In the next few sections, when we construct long sequences of these graphs, tokens will be moved between adjacent processors within a round, and used to guarantee that processor failures in adjacent graphs change in a orderly fashion. For every value of ℓ, we define graphs with exactly ℓ tokens placed on nodes in each round, but we will be most interested in the two cases with ℓ equal to 1 and k.

For each value $\ell > 0$, we define an *ℓ-graph* $\mathcal{G}$ to be a communication graph with tokens placed on the nodes of the graph that satisfies the following conditions for each round i, $1 \leq i \leq r$:

1. The total number of tokens on round i nodes is exactly ℓ.
2. If a round i edge from p is red, then there is a token on a round $j \leq i$ node for p.
3. If a round i edge from p is red, then p is silent from round $i+1$.

We say that p is *covered by a round i token* if there is a token on the round i node for p, we say that p is *covered in round i* if p is covered by a round $j \leq i$ token, and we say that p is *covered* in a graph if p is covered in any round. Similarly, we say that a round i edge from p is covered if p is covered in round i. The second condition says every red edge is covered by a token, and this together with the first condition implies that at most ℓr processors fail in an ℓ-graph. We often refer to an ℓ-graph as a *graph* when the value of ℓ is clear from context or unimportant. We emphasize that the tokens are simply an accounting trick, and have no meaning as part of the global or local state in the underlying communication graph.

We define a *failure-free* ℓ-graph to be an ℓ-graph in which all edges are green, and all round i tokens are on processor p_1 in all rounds i.

6.2 Graph operations

We now define four operations on augmented graphs that make only minor changes to a graph. In particular, the only change an operation makes is to change the color of a single edge, to change the value of a single processor's input, or to move a single token between adjacent processors within the same round. The operations are defined as follows:

1. *delete*(i,p,q): This operation changes the color of the round i edge from p to q to red, and has no effect if the edge is already red. This makes the delivery of the round i message from p to q unsuccessful. It can only be applied to a graph if p and q are silent from round $i+1$, and p is covered in round i.
2. *add*(i,p,q): This operation changes the color of the round i edge from p to q to green, and has no effect if the edge is already green. This makes the delivery of the round i message from p to q successful. It can only be applied to a graph if p and q are silent from round $i+1$, processor p is active through round $i-1$, and p is covered in round i.
3. *change*(p,v): This operation changes the input value for processor p to v, and has no effect if the value is already v. It can only be applied to a graph if p is silent from round 1, and p is covered in round 1.
4. *move*(i,p,q): This operation moves a round i token from $\langle p, i-1\rangle$ to $\langle q, i-1\rangle$, and is defined only for adjacent processors p and q (that is, $\{p,q\} = \{p_j, p_{j+1}\}$ for some j). It can only be applied to a graph if p is covered by a round i token, and all red edges are covered by other tokens.

It is obvious from the definition of these operations that they preserve the property of being an ℓ-graph: if $\mathcal{G}$ is an ℓ-graph and τ is a graph operation, then $\tau(\mathcal{G})$ is an ℓ-graph. We define *delete*, *add*, and *change* operations on communication graphs in exactly the same way, except that the condition "p is covered in round i" is omitted.

6.3 Graph sequences

We now define a sequence $\sigma[v]$ of graph operations that can be applied to any failure-free graph $\mathcal{G}$ to transform it into the failure-free graph $\mathcal{G}[v]$ in which all processors have input v. We want to emphasize that the sequences $\sigma[v]$ differ only in the value v. For this reason, we define a *parameterized sequence* $\sigma[\mathrm{x}_1, \ldots, \mathrm{x}_\ell]$ to be a sequence of graph operations with free variables $\mathrm{x}_1, \ldots, \mathrm{x}_\ell$ appearing as parameters to the graph operations in the sequence.

Given a graph $\mathcal{G}$, let $\mathcal{G}_i[v]$ be the graph identical to $\mathcal{G}$, except that processor p_i has input v. In the case of ordinary communication graphs, a result by Moses and Tuttle [MT88] implies that if $\mathcal{G}$ and $\mathcal{G}_i[v]$ are failure-free graphs, then there is a "similarity chain" of graphs between $\mathcal{G}$ and $\mathcal{G}_i[v]$. In their proof—a refinement of similar proofs by Dwork and Moses [DM90] and others—the sequence of graphs

they construct has the property that each graph in the chain can be obtained from the preceding graph by applying a sequence of the *add*, *delete*, and *change* graph operations defined above. The same proof works for augmented communication graphs, provided we insert *move* operations between the *add*, *delete*, and *change* operations to move tokens between nodes appropriately. With this simple modification, we can prove the following.

Lemma 1: For each i, there is a parameterized sequence $\sigma_i[\mathrm{v}]$ with the property that for all values v and failure-free graphs $\mathcal{G}$, the sequence $\sigma_i[v]$ transforms $\mathcal{G}$ to $\mathcal{G}_i[v]$.

By concatenating such operation sequences, we can transform $\mathcal{G}$ into $\mathcal{G}[v]$ by changing processors' input values one at a time:

Lemma 2: Let $\sigma[\mathrm{v}] = \sigma_1[\mathrm{v}]\cdots\sigma_n[\mathrm{v}]$. For every value v and every failure-free graph $\mathcal{G}$, the sequence $\sigma[v]$ transforms $\mathcal{G}$ to $\mathcal{G}[v]$.

Now we can define the parameter N used in defining the shape of B: N is the length of the sequence $\sigma[\mathrm{v}]$, which is exponential in r.

6.4 Graph merge

Speaking informally, we will use each sequence $\sigma[v_i]$ of graph operations to generate a sequence of graphs, and we will use this sequence of graphs to label vertices along the edge of B in the ith dimension. Then we will label vertices in the interior of B by performing a "merge" of the graphs on the edges in the different dimensions.

The *merge* of a sequence $\mathcal{H}_1, \ldots, \mathcal{H}_k$ of graphs is a graph defined as follows:

1. an edge e is colored red if it is red in any of the graphs $\mathcal{H}_1, \ldots, \mathcal{H}_k$, and green otherwise, and
2. an initial node $\langle p, 0\rangle$ is labeled with the value v_i where i is the maximum index such that $\langle p, 0\rangle$ is labeled with v_i in $\mathcal{H}_i$, or v_0 if no such i exists, and
3. the number of tokens on a node $\langle p, i\rangle$ is the sum of the number of tokens on the node in the graphs $\mathcal{H}_1, \ldots, \mathcal{H}_k$.

The first condition says that a message is missing in the resulting graph if and only if it is missing in any of the merged graphs. To understand the second condition, notice that for each processor p_j there is a integer s_j with the property that p_j's input value in changed to v_i by the s_jth operation appearing in $\sigma[v_i]$. Now choose a vertex $x = (x_1, \ldots, x_k)$ of B, and imagine walking from the origin to x by walking along the first dimension to $(x_1, 0, \ldots, 0)$, then along the second dimension to $(x_1, x_2, 0, \ldots, 0)$, and so forth. In each dimension i, processor p_j's input is changed from v_{i-1} to v_i after s_j steps in this dimension. Since $x_1 \geq x_2 \geq \cdots \geq x_k$, there is a final dimension i in which p_j's input is changed to v_i, and never changed again. The second condition above is just a compact way of identifying this final value v_i.

Lemma 3: Let $\mathcal{H}$ be the merge of the graphs $\mathcal{H}_1, \ldots, \mathcal{H}_k$. If $\mathcal{H}_1, \ldots, \mathcal{H}_k$ are 1-graphs, then $\mathcal{H}$ is a k-graph.

6.5 Graph assignments

Now we can define the assignment of graphs to vertices of B. For each value v_i, let $\mathcal{F}_i$ be the failure-free 1-graph in which all processors have input v_i. Let $x = (x_1, \ldots, x_k)$ be an arbitrary vertex of B. For each coordinate x_j, let σ_j be the prefix of $\sigma[v_j]$ consisting of the first x_j operations, and let $\mathcal{H}_j$ be the 1-graph resulting from the application of σ_j to $\mathcal{F}_{j-1}$. This means that in $\mathcal{H}_j$, some set $p_1, \ldots, p_i$ of adjacent processors have had their inputs changed from v_{j-1} to v_j. The graph $\mathcal{G}$ labeling x is defined to be the merge of $\mathcal{H}_1, \ldots, \mathcal{H}_k$. We know that $\mathcal{G}$ is a k-graph by Lemma 3, and hence that at most $rk \leq f$ processors fail in $\mathcal{G}$.

Remember that we always write the vertices of a primitive simplex in a canonical order $y_0, \ldots, y_k$. In the same way, we always write the graphs labeling the vertices of a primitive simplex in the canonical order $\mathcal{G}_0, \ldots, \mathcal{G}_k$, where $\mathcal{G}_i$ is the graph labeling y_i.

6.6 Graphs on a simplex

The graphs labeling the vertices of a primitive simplex have some convenient properties. For this section, fix a primitive simplex S, and let $y_0, \ldots, y_k$ be the vertices of S and let $\mathcal{G}_0, \ldots, \mathcal{G}_k$ be the graphs labeling the corresponding vertices of S. Our first result says that any processor that is uncovered at a vertex of S is nonfaulty at all vertices of S.

Lemma 4: If processor q is not covered in the graph labeling a vertex of S, then q is nonfaulty in the graph labeling every vertex of S.

Our next result shows that we can use the bound on the number of tokens to bound the number of processors failing at any vertex of S.

Lemma 5: If F_i is the set of processors failing in $\mathcal{G}_i$ and $F = \cup_i F_i$, then $|F| \leq rk \leq f$.

We have assigned graphs to S, and now we must assign processors to S. A *local processor labeling* of S is an assignment of distinct processors $q_0, \ldots, q_k$ to the vertices $y_0, \ldots, y_k$ of S so that q_i is uncovered in $\mathcal{G}_i$ for each y_i. A *global processor labeling* of B is an assignment of processors to vertices of B that induces a local processor labeling at each primitive simplex. The final important property of the graphs labeling S is that if we use a processor labeling to label S with processors, then S is consistent with a single global communication graph. The proof of this requires a few preliminary results.

Lemma 6: If $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$ differ in p's input value, then p is silent from round 1 in $\mathcal{G}_0, \ldots, \mathcal{G}_k$. If $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$ differ in the color of an edge from p to q in round t, then p and q are silent from round $t+1$ in $\mathcal{G}_0, \ldots, \mathcal{G}_k$.

Lemma 7: If $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$ differ in the local communication graph of p at time t, then p is silent from round $t+1$ in $\mathcal{G}_0, \ldots, \mathcal{G}_k$.

Lemma 8: If p sends a message in round r in any of the graphs $\mathcal{G}_0, \ldots, \mathcal{G}_k$, then p has the same local communication graph at time $r-1$ in all of the graphs $\mathcal{G}_0, \ldots, \mathcal{G}_k$.

Finally, we can prove the crucial property of primitive simplexes in the Bermuda Triangle:

Lemma 9: Given a local processor labeling of S, let $q_0, \ldots, q_k$ be the processors labeling the vertices of S, and let $\mathcal{L}_i$ be the local communication graph of q_i in $\mathcal{G}_i$. There is a global communication graph $\mathcal{G}$ with the property that each q_i is nonfaulty in $\mathcal{G}$ and has the local communication graph $\mathcal{L}_i$ in $\mathcal{G}$.

7 Processor Assignment

What Lemma 9 at the end of the preceding section tells us is that all we have left to do is to construct a global processor labeling. In this section, we show how to do this. We first associate a set of "live" processors with each communication graph labeling a vertex of B, and then we choose one processor from each set to label vertices of B.

7.1 Live processors

Given a graph $\mathcal{G}$, we construct a set of $c = n - rk \geq k+1$ uncovered (and hence nonfaulty) processors. We refer to these processors as the *live* processors in $\mathcal{G}$, and we denote this set by $live(\mathcal{G})$. These live sets have one crucial property: if $\mathcal{G}$ and $\mathcal{G}'$ are two graphs labeling adjacent vertices, and if p is in both $live(\mathcal{G})$ and $live(\mathcal{G}')$, then p has the same rank in both sets. As usual, we define the *rank* of p_i in a set R of processors to be the number of processors $p_j \in R$ with $j \leq i$.

```
S ← {1, ..., n}
for each i = 1, ..., n
  count ← 0
  for each j = i, i − 1, ..., 1, i + 1, ..., n
    if count = m_i then break
    if j ∈ S then
      S ← S − {j}
      count ← count + 1
live(G) ← S
```

Figure 3: The construction of $live(\mathcal{G})$.

Given a graph $\mathcal{G}$, we now show how to construct $live(\mathcal{G})$. This construction has one goal: if $\mathcal{G}$ and $\mathcal{G}'$ are graphs labeling adjacent vertices, then the construction should minimize the number of processors whose rank differs in the sets $live(\mathcal{G})$ and $live(\mathcal{G}')$. The construction of $live(\mathcal{G})$ begins with the set of all processors, and removes a set of rk processors, one for each token. This set of removed processors includes the covered processors, but may include other processors as well. For example, suppose p_i and p_{i+1} are covered with one token each in $\mathcal{G}$, but suppose p_i is uncovered and p_{i+1} is covered by two tokens in $\mathcal{G}'$. For simplicity, let's assume these are the only tokens on the graphs. When constructing the set $live(\mathcal{G})$, we remove both p_i and p_{i+1} since they are both covered. When constructing the set $live(\mathcal{G}')$, we remove p_{i+1}, but we must also remove a second processor corresponding to the second token covering p_{i+1}. Which processor should we remove? Notice that if we choose to remove p_i again, then no processors change rank. In general, the construction of $live(\mathcal{G})$ considers each processor p in turn. If p is covered by m_p tokens in $\mathcal{G}$, then the construction removes m_p processors by starting with p, working down the list of remaining processors smaller than p, and then working up the list of processors larger than p if necessary.

Specifically, given a graph $\mathcal{G}$, the *multiplicity* of p is the number m_p of tokens appearing on nodes for p in $\mathcal{G}$, and the *multiplicity* of $\mathcal{G}$ is the vector $m = \langle m_{p_1}, \ldots, m_{p_n} \rangle$. Given the multiplicity of $\mathcal{G}$ as input, the algorithm in Figure 3 computes $live(\mathcal{G})$. In this algorithm, processor p_i is denoted by its index i. This construction has two obvious properties: If $i \in live(\mathcal{G})$ then $m_i = 0$ (i is uncovered),

and $\sum_{j=1}^{i-1} m_j \leq i-1$.

The assignment of graphs to the corners of a simplex has the property that once p becomes covered on one corner of S, it remains covered on the following corners of S:

Lemma 10: If p is uncovered in the graphs $\mathcal{G}_i$ and $\mathcal{G}_j$, where $i < j$, then p is uncovered in each graph $\mathcal{G}_i, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_j$.

Finally, because token placements in adjacent graphs on a simplex differ in at most the movement of one token from one processor to an adjacent processor, we can use the preceding lemma to prove the following:

Lemma 11: If $p \in \mathit{live}(\mathcal{G}_i)$ and $p \in \mathit{live}(\mathcal{G}_j)$, then p has the same rank in $\mathit{live}(\mathcal{G}_i)$ and $\mathit{live}(\mathcal{G}_j)$.

7.2 Processor labeling

We now choose one processor from each set $\mathit{live}(\mathcal{G})$ to label the vertex with graph $\mathcal{G}$. Given a vertex $x = (x_1, \ldots, x_k)$, we define $\mathit{plane}(x) = \sum_{i=1}^{k} x_i \pmod{k+1}$.

Lemma 12: If x and y are distinct vertices of the same simplex, then $\mathit{plane}(x) \neq \mathit{plane}(y)$.

We define a global processor labeling π as follows: given a vertex x labeled with a graph $\mathcal{G}$, we define π to map x to the processor having rank $\mathit{plane}(x)$ in $\mathit{live}(\mathcal{G})$.

Lemma 13: The mapping π is a global processor labeling.

We label the vertices of B with processors according to the processor labeling π.

8 Ordered Pair Assignment

Finally, we assign ordered pairs $(p, \mathcal{L})$ of processor ids and local communication graphs to vertices of B. Given a vertex x labeled with processor p and graph $\mathcal{G}$, we label x with the ordered pair $(p, \mathcal{L})$ where $\mathcal{L}$ is the local communication graph of p in $\mathcal{G}$. The following result is a direct consequence of Lemmas 9 and 13. It says that the local communication graphs of processors labeling the corners of a vertex are consistent with a single global communication graph.

Lemma 14: Let $q_0, \ldots, q_k$ and $\mathcal{L}_0, \ldots, \mathcal{L}_k$ be the processors and local communication graphs labeling the vertices of a simplex. There is a global communication graph $\mathcal{G}$ with the property that each q_i is nonfaulty in $\mathcal{G}$ and has the local communication graph $\mathcal{L}_i$ in $\mathcal{G}$.

9 Sperner's Lemma

We now state Sperner's Lemma, and use it to prove a lower bound on the number of rounds required to solve k-set agreement.

Informally, a Sperner coloring of B assigns a color to each vertex so that each corner vertex c_i is given a distinct color w_i, each vertex on the edge between c_i and c_j is given either w_i or w_j, and so on. More formally, let S be a simplex and let F be a face of S. Any triangulation of S induces a triangulation of F in the obvious way. Let T be a triangulation of S. A *Sperner coloring* of T assigns a color to each vertex of T so that each corner of T has a distinct color, and so that the vertices contained in a face F are colored with the colors on the corners of F, for each face F of T. Sperner colorings have a remarkable property: at least one simplex in the triangulation must be given all possible colors.

Lemma 15 (Sperner's Lemma): If B is a triangulation of a k-simplex, then for any Sperner coloring of B, there exists at least one k-simplex in B whose vertices are all given distinct colors.

Let P be the protocol whose existence we assumed in the previous section. Define a coloring χ_P of B as follows. Given a vertex x labeled with processor p and local communication graph $\mathcal{L}$, color x with the value v that P requires processor p to choose when its local communication graph is $\mathcal{L}$. This coloring is clearly well-defined, since P is a protocol in which all processors chose an output value at the end of round r. Formalizing the argument sketched in the introduction, we can show that χ_P is a Sperner coloring.

Lemma 16: If P is a protocol for k-set agreement tolerating f faults and halting in $r \leq \lfloor f/k \rfloor$ rounds, then χ_P is a Sperner coloring of B.

Consequently, we can use Sperner's Lemma to prove that there exists a global state in which $k+1$ processors choose $k+1$ distinct values.

Theorem 17: If $n \geq f + k + 1$, then no protocol for k-set agreement can halt in fewer than $\lfloor f/k \rfloor + 1$ rounds.

Technical Report: A full version of this work will soon be available for electronic distribution. Send the one-word message "help" to techreports@crl.dec.com for information.

Acknowledgements: This work was performed while the first author was visiting MIT. The first and third authors were supported in part by NSF grant CCR-89-15206, in part by DARPA contracts N00014-89-J-1988, N00014-92-J-4033, and N00014-92-J-1799, and in part by ONR contract N00014-91-J-1046.

References

[BHG87] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1987.

[Cha91] Soma Chaudhuri. Towards a complexity hierarchy of wait-free concurrent objects. In *Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing.* IEEE, December 1991. Also appeared as Technical Report No. 91-024, Iowa State University, 1991.

[Cha93] Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105:132–158, July 1993. A preliminary version appeared in ACM PODC, 1990.

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

[Dol82] Danny Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, March 1982.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(3):656–666, November 1983.

[Fis83] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In Marek Karpinsky, editor, *Proceedings of the 10th International Colloquium on Automata, Languages, and Programming*, pages 127–140. Springer-Verlag, 1983. A preliminary version appeared as Yale Technical Report YALEU/DCS/RR-273.

[FL82] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, June 1982.

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

[Had83] Vassos Hadzilacos. A lower bound for Byzantine agreement with fail-stop processors. Technical Report TR–21–83, Harvard University, 1983.

[Her91] Maurice P. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124–149, January 1991.

[HS93] Maurice P. Herlihy and Nir Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 111–120. ACM, May 1993.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[Mer85] Michael Merritt. Notes on the Dolev-Strong lower bound for byzantine agreement. Unpublished manuscript, 1985.

[MT88] Yoram Moses and Mark R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3(1):121–169, 1988.

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

[Spa66] E.H. Spanier. *Algebraic Topology.* Springer-Verlag, New York, 1966.

[W+78] J. H. Wensley et al. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, October 1978.