Tight Bounds for k-Set Agreement

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

January 21, 2000

Abstract

We prove tight bounds on the time needed to solve *k-set agreement*. In this problem, each processor starts with an arbitrary input value taken from a fixed set, and halts after choosing an output value. In every execution, at most k distinct output values may be chosen, and every processor's output value must be some processor's input value. 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, and we exhibit a protocol proving the matching upper bound. This result 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 the first to apply topological techniques to the synchronous model.

*Department of Computer Science, Iowa State University, Ames, IA 50011-1040; chaudhur@cs.iastate.edu.

†Department of Computer Science, Brown University, Providence, RI 02912; herlihy@cs.brown.edu.

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

§Compaq Computer Corporation, Cambridge Research Laboratory, One Kendall Square, Building 700, Cambridge, MA 02139; tuttle@crl.dec.com.

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.

The *k-set agreement problem* [Cha93] is defined as follows. Each processor has a read-only input register and a write-only output register. Each processor begins with an arbitrary input value in its input register from a set V containing at least $k+1$ values $v_0, \ldots, v_k$, and nothing in its output register. A protocol solves k-set agreement if, in every execution, the nonfaulty processors halt after writing output values to their output registers that satisfy two conditions:

1. *validity*: every processor's output value is some processor's input value, and
2. *agreement:* 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 hardwired 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 simply chooses 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. Consensus arises in applications as diverse as on-board aircraft control [W$^+$78], database transaction commit [BHG87], and concurrent object design [Her91]. 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

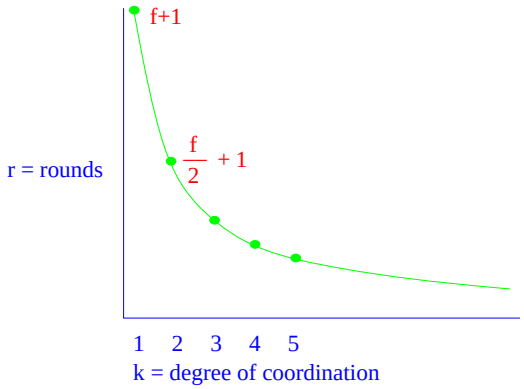

Figure 1: Tradeoff between rounds and degree of coordination.

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 crashing in the middle of a round. When a processor crashes, it sends some subset of the messages it is required to send in that round by the protocol, and then sends no messages in any later round.

The primary contribution of this paper is a tight bound on the amount of time required to solve k-set agreement. We prove that any protocol solving k-set agreement requires $\lfloor f/k \rfloor + 1$ rounds of communication, where f is the bound on the number of processors allowed to fail in any execution of the protocol, and we give a protocol that solves k-set agreement in this number of rounds, proving that this bound is tight. Since consensus is just 1-set agreement, our lower bound implies the well-known lower bound of $f + 1$ rounds for solving consensus [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. This tradeoff is illustrated in Figure 1 for a fixed value of f, and it shows, for example, that 2-set agreement can be achieved in half the time needed to achieve consensus. The proof of the lower bound is an interesting geometric proof that combines ideas due to Chaudhuri [Cha91, Cha93], Fischer and Lynch [FL82], Herlihy and Shavit [HS99], and Dwork, Moses, and Tuttle [DM90, MT88].

Of these ideas, the notion that concepts from topology can be used to analyze concurrent systems has received a considerable amount of attention. In the past few years, researchers have developed powerful new tools based on classical algebraic topology for analyzing tasks in asynchronous models [AR96, BG93, GK99, HR94, HR95, HS99, SZ93]. The principal innovation of these papers is to model computations as simplicial complexes (rather than graphs) and to derive connections between computations and the topological properties of these complexes. Our paper extends this topological approach in several new ways: it is the first to derive results in the synchronous model, it derives lower bounds rather than computability results, and it uses explicit constructions instead of existential arguments. For example, our paper is closely related to the work of Herlihy and Shavit [HS99] and their Asynchronous Computability Theorem. They work in the asynchronous model, they prove the existence of a simplicial complex representing the global states of a system at the end of a protocol, and they prove that this protocol solves k-set agreement if and only if this simplicial complex is not too highly connected. In contrast, we work in the synchronous model, and it is not at all clear how to translate their results into this model. In addition, our proof is constructive and not existential, in the sense that we explicitly construct a small portion of what would be the synchronous simplicial complex containing a state violating the requirements of k-set agreement, and hence proving the lower bound. Finally, we prove a lower bound on time complexity and not an impossibility result.

Although the synchronous model makes some strong (and possibly unrealistic) assumptions, it is well-suited for proving lower bounds. The synchronous model is a special case of almost every realistic model of a concurrent system we can imagine, and therefore any lower bound for k-set agreement in this simple model translates into a lower bound in any more complex model. For example, our lower bound holds for models that permit messages to be lost, failed processors to restart, or processor speeds to vary. Moreover, our techniques may be helpful in understanding how to prove (possibly) stricter lower bounds in more complex models. Naturally, our protocol for k-set agreement in the synchronous model does not work in more general models, but it is still useful because it shows that our lower bound is the best possible in the synchronous model.

This paper is organized as follows. In Section 2 we give an optimal protocol for k-set agreement, establishing our upper bound for the problem. In Section 3, we give an informal overview of our matching lower bound, in Section 4 we define our model of computation, and in Sections 5 through 9 we prove our lower bound, proving that our bound is tight.

best $\leftarrow$ *input_value*;

for each round 1 through $\lfloor f/k \rfloor + 1$ do
  broadcast *best*;
  receive values $b_1, \ldots, b_\ell$ from other processors;
  best $\leftarrow \min\{b_1, \ldots, b_\ell\}$;

choose *best*.

Figure 2: An optimal protocol P for k-set agreement.

2 An optimal protocol for k-set agreement

The protocol P given in Figure 2 is an optimal protocol for the k-set agreement problem. In this protocol, processors repeatedly broadcast input values and keep track of the least input value received in a local variable *best*. Initially, a processor sets *best* to its own input value. In each of the next $\lfloor f/k \rfloor + 1$ rounds, the processor broadcasts the value of *best* and then sets *best* to the smallest value received in that round from any processor (including itself). In the end, it chooses the value of *best* as its output value.

To prove that P is an optimal protocol, we must prove that, in every execution of P, processors halt in $r = \lfloor f/k \rfloor + 1$ rounds, every processor's output value is some processor's input value, and the set of output values chosen has size at most k. The first two statements follow immediately from the text of the protocol, so we need only prove the third. For each time t and processor p, let $best_{p,t}$ be the value of *best* held by p at time t. For each time t, let $Best(t)$ be the set of values $best_{q_1,t}, \ldots, best_{q_\ell,t}$ where the processors $q_1, \ldots, q_\ell$ are the processors active through round t, by which we mean that they send all messages required by the protocol in all rounds through the end of round t. Notice that $Best(0)$ is the set of input values, and that $Best(r)$ is the set of chosen output values. Our first observation is that the set $Best(t)$ never increases from one round to the next.

Lemma 1: $Best(t) \supseteq Best(t+1)$ for all times t.

Proof: If $b \in Best(t+1)$, then $b = best_{p,t+1}$ for some processor p active through round $t+1$. Since $best_{p,t+1}$ is the minimum of the values $b_1, \ldots, b_\ell$ sent to p by processors during round $t+1$, we know that $b = best_{q,t}$ for some processor q that is active through round t. Consequently, $b \in Best(t)$. □

We can use this observation to prove that the only executions in which many output values are chosen are executions in which many processors fail. We say that a processor p *fails before time* t if there is a processor q to which p sends no message in round t (and p may fail to send to q in earlier rounds as well).

Lemma 2: If $|Best(t)| = d+1$, then at least dt processors fail before time t.

Proof: We proceed by induction on t. The case of $t = 0$ is immediate, so suppose that $t > 0$ and that the induction hypothesis holds for $t - 1$. Since $|Best(t)| = d + 1$ and since $Best(t) \subseteq Best(t - 1)$ by Lemma 1, it follows that $|Best(t - 1)| \geq d + 1$, and the induction hypothesis for $t - 1$ implies that there is a set S of $d(t-1)$ processors that fail before time $t-1$. It is enough to show that there are an additional d processors not contained in S that fail before time t.

Let $b_0, \ldots, b_d$ be the values of $Best(t)$ written in increasing order. Let q be a processor with $best_{q,t}$ set to the largest value b_d at time t, and for each value b_i let q_i be a processor that sent b_i in round $t - 1$. The processors $q_0, \ldots, q_d$ are distinct since the values $b_0, \ldots, b_d$ are distinct, and these processors do not fail before time $t-1$ since they send a message in round t, so they are not contained in S. On the other hand, the processors $q_0, \ldots, q_{d-1}$ sending the small values $b_0, \ldots, b_{d-1}$ in round $t-1$ clearly did not send their values to the processor q setting $best_{q,t}$ to the large value b_d, or q would have set $best_{q,t}$ to a smaller value. Consequently, these d processors $q_0, \ldots, q_{d-1}$ fail in round t and hence fail before time t. □

Since $Best(r)$ is the set of output values chosen by processors at the end of round $r = \lfloor f/k \rfloor + 1$, if $k + 1$ output values are chosen, then Lemma 2 says that at least kr processors fail, which is impossible since $f < kr$. Consequently, the set of output values chosen has size at most k, and we are done.

Theorem 3: The protocol P solves k-set agreement in $\lfloor f/k \rfloor + 1$ rounds.

The proof of Lemma 2 gives us some idea of how long chains of processor failures can force the protocol to run for $\lfloor f/k \rfloor + 1$ rounds, and we have illustrated one of these long executions in Figure 3, where $k = 2$ and $f = 4$ and the protocol must run for $\lfloor f/k \rfloor + 1 = 3$ rounds. In this figure, the processor ids and initial inputs are indicated on the left, time is indicated on the bottom, and each node representing the state of a processor q at a time t is labeled with the value $best_{q,t}$ in that state. In this figure, we

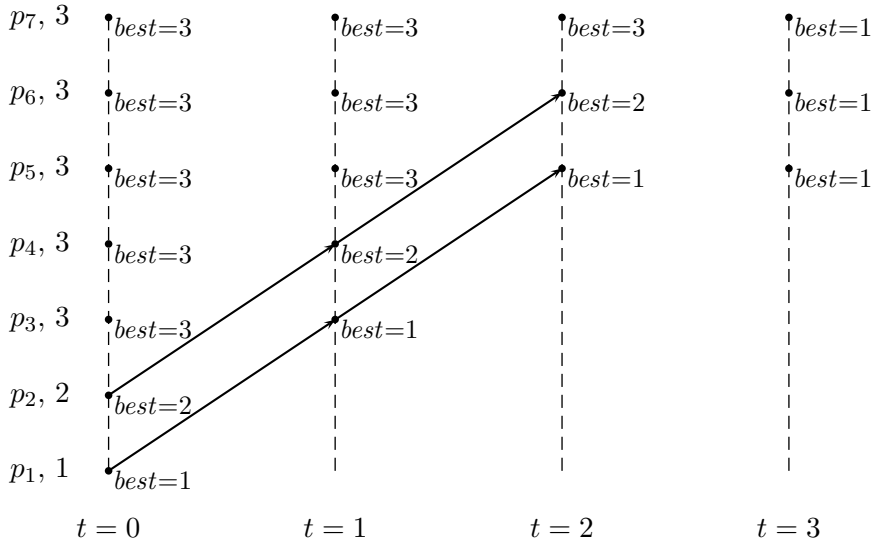

Figure 3: An illustration of how chains of processor failures can force the optimal protocol for k-set agreement to run for $\lfloor f/k \rfloor + 1$ rounds, where $k = 2$ and $f = 4$.

illustrate only the messages sent by faulty processors during their failing rounds, and leave implicit the fact that processors broadcast messages to all other processors in all other rounds before they crash.

In this execution, all processors begin with input value 3, except that processors p_1 and p_2 start with 1 and 2. Initially, each processor q sets $best_{q,0}$ to its input value. Processors p_1 and p_2 crash during round one and send their values 1 and 2 only to processors p_3 and p_4, respectively. At the end of the first round, all processors q set $best_{q,1}$ to 3, except for p_3 and p_4 that set $best_{q,1}$ to 1 and 2. Processors p_3 and p_4 crash during round two and send their values 1 and 2 only to processors p_5 and p_6. If the processors were to halt at the end of the second round, they would end up choosing three distinct values instead of two, since the two failure chains have effectively hidden the two input values 1 and 2 from the last processor p_7. Since the processors are supposed to be solving k-set agreement for $k = 2$, the processors must continue for one more round, in which all processors will broadcast to each other (no processor can fail in the last round since all $f = 4$ failures have occurred) and choose 1 as their final value.

With this example in mind, turning our attention to the lower bound, if an adversary controlling the failure of processors were to try to disrupt

a k-set agreement protocol by using f failures to hide k of the input values, how long could the adversary keep these values hidden? It is clear that the adversary can maintain k distinct failure chains for at most f/k rounds, since it must fail k processors each round, allowing the protocol to halt after just $f/k + 1$ rounds. With this intuition, we now turn our attention to proving that any protocol for k-set agreement requires $\lfloor f/k \rfloor + 1$ rounds of communication, assuming $n \geq f + k + 1$ to rule out the degenerate case where k or fewer processors remain in the last round and can solve k-set agreement simply by choosing any of the input values as their output value.

3 An overview of the lower bound proof

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 involves showing that two time r global states of P, one in which all processors start with 0 and one in which all processors start with 1, lie on a single similarity chain. Since there is a similarity chain from one state to the other, processors must choose the same value in both states, 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, focusing on the number and identity of local states common to two global states. While this seems difficult—if not impossible—to do using conventional graph theoretic techniques like similarity chains, there is a *geometric* generalization of similarity chains that provides a compact way of capturing all degrees of similarity simultaneously, and it is the basis of our proof.

A simplex is just the natural generalization of a triangle to n 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. We can represent a global state for an n-processor protocol as an $(n-1)$-dimensional simplex [Cha93, HS99], where each vertex is labeled with a processor id and local state. If g_1 and g_2 are global states in which p_1 has the same local state, then we "glue together" the vertices of g_1 and g_2 labeled with p_1. Figure 4 shows how these global states glue together in a simple protocol in which each of three processors repeatedly sends its state to the others. Each process begins with a binary input. The first picture shows the possible global states after zero rounds: since no communication has occurred, each processor's state consists only of its input. It is easy to check that the simplexes corresponding to these global states form an octahedron. The next picture shows the complex after one round. Each triangle corresponds to a failure-free execution, each free-standing edge to a single-failure execution, and so on. The third picture shows the possible global states after three rounds.

The set of global states after an r-round protocol is quite complicated

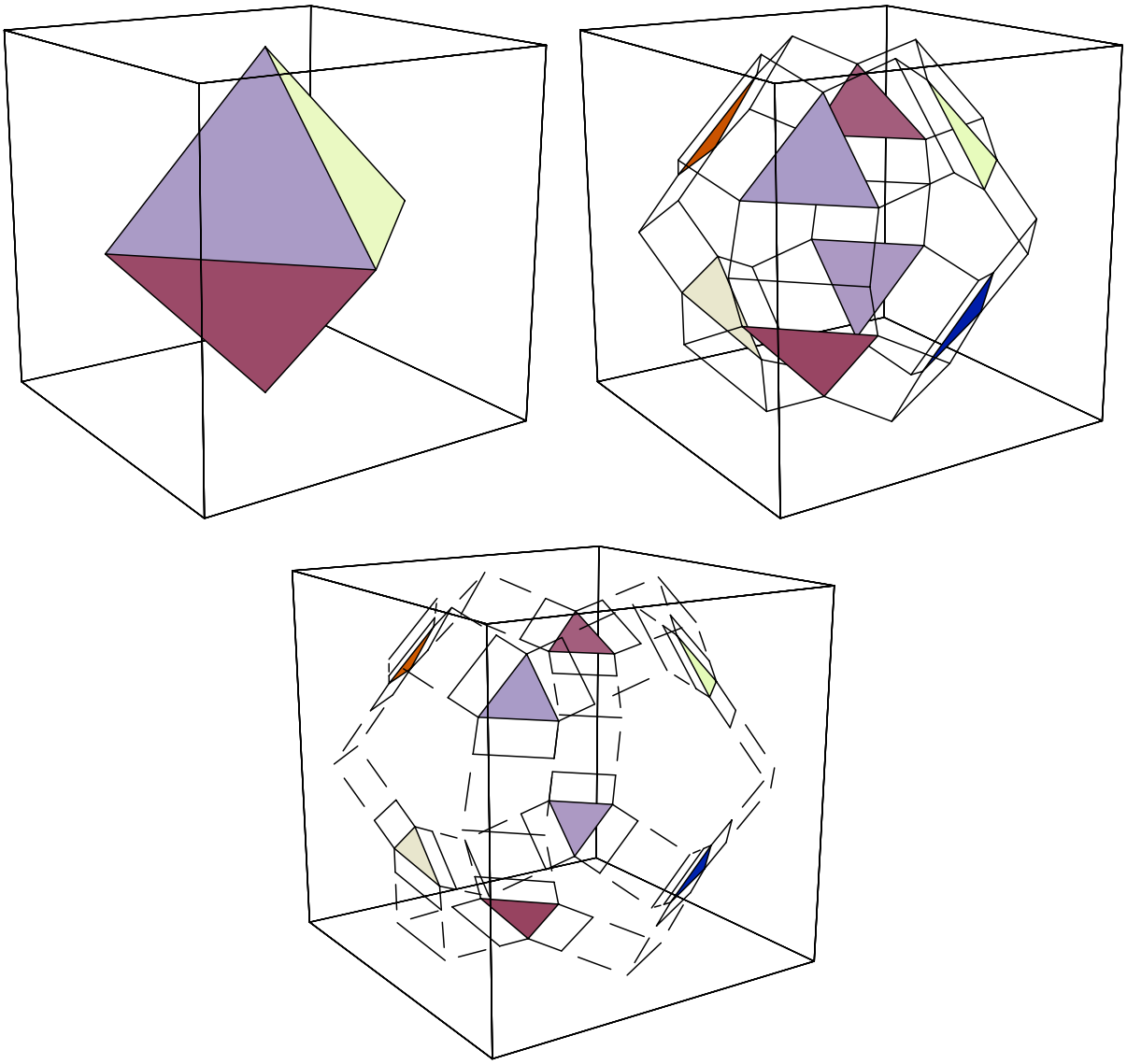

Figure 4: Global states for zero, one, and two-round protocols.

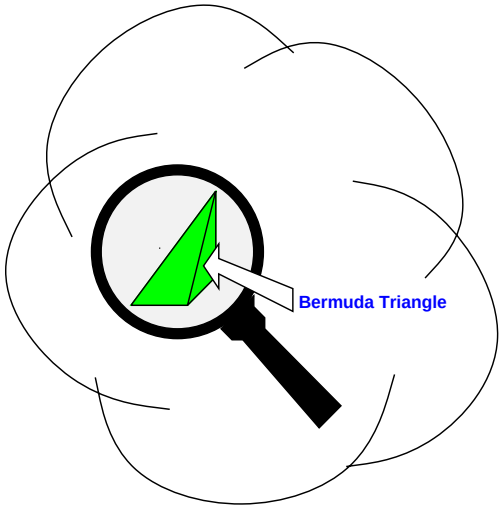

Figure 5: The Bermuda Triangle is a highly-structured subcomplex of the simplicial complex representing all global states of an r-round protocol, such as the simplicial complexes illustrated in Figure 4 for the cases of zero, one, and two round protocols.

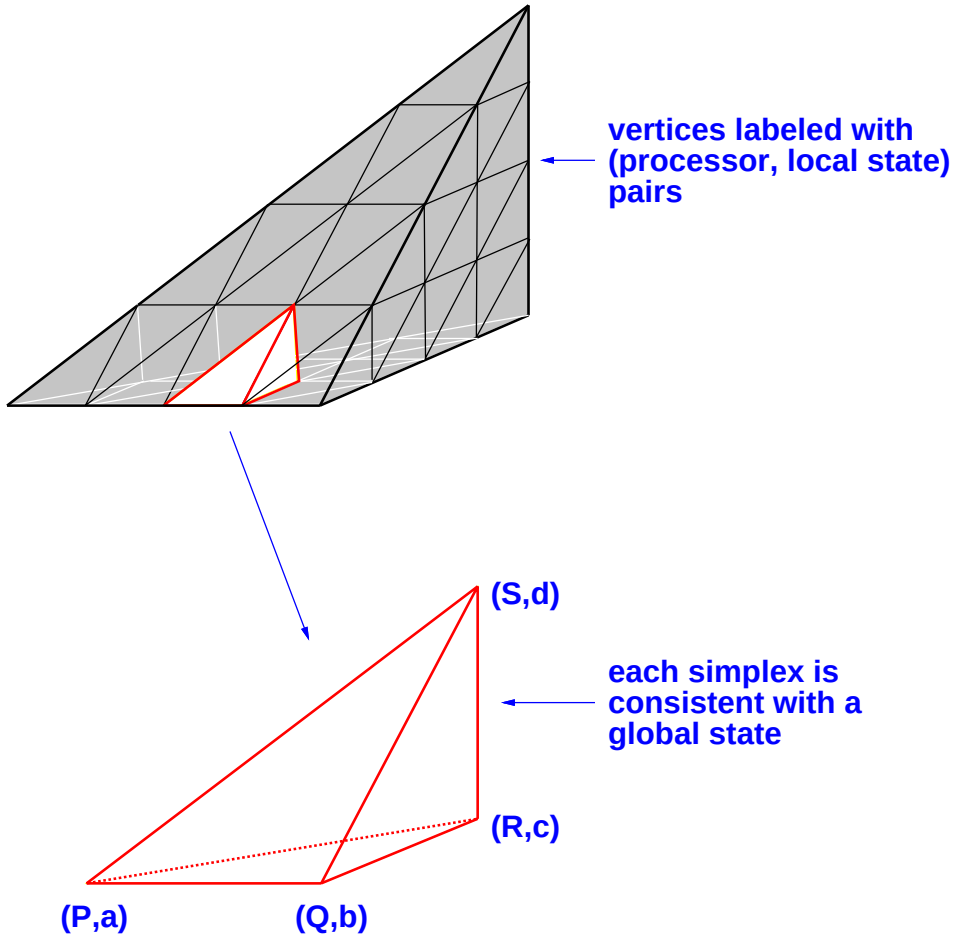

Figure 6: Bermuda Triangle with simplex representing typical global state.

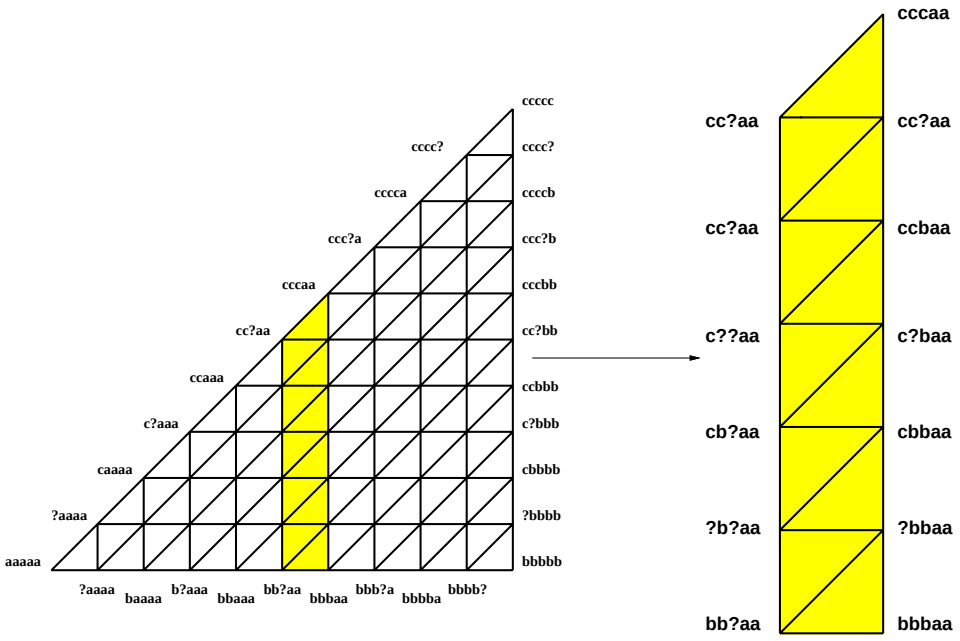

Figure 7: A simplified representation of the Bermuda Triangle for 5 processors and a 1-round protocol for 2-set agreement. This representation omits the processor ids labeling the vertices to focus on how the local states change along each dimension of the Bermuda Triangle.

(Figure 5), but it contains a well-behaved subset of global states which we call the *Bermuda Triangle* B, since all fast protocols vanish somewhere in its interior. The Bermuda Triangle (Figure 6) is constructed by starting with a large k-dimensional simplex, and *triangulating* it into a collection of smaller k-dimensional simplexes. We then label each vertex 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.

To illustrate the process of labeling vertices, Figure 7 shows a simplified representation of a two-dimensional Bermuda Triangle B. It is the Bermuda Triangle for a protocol P 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 val-

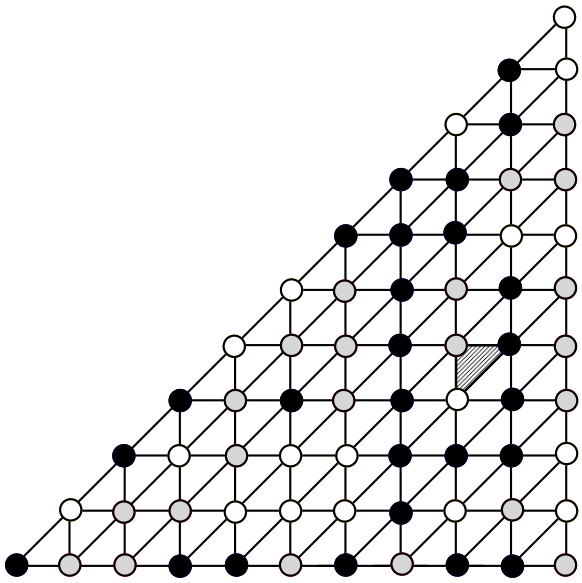

Figure 8: Sperner's Lemma.

ues 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 7, 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 shown in Figure 7—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 (Figure 8). 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 basic structure of the Bermuda Triangle and the idea of coloring the vertices with decision values and applying Sperner's Lemma have appeared in previous work by Chaudhuri [Cha91, Cha93]. In that work, she also proved a lower bound of $\lfloor f/k \rfloor + 1$ rounds for k-set agreement, but for a very restricted class of protocols. In particular, a protocol's decision function can depend only on vectors giving partial information about which processors started with which input values, but cannot depend on any other information in a processor's local state, such as processor identities or message histories. The proof of her lower bound depends on this restriction to a specific class of protocols. 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 k-set agreement for arbitrary protocols.

Our approach consists of four parts. First, we construct long sequences of global states that we call *similarity chains*, one chain for each pair of input values. For example, for the pair of values a and b, we construct a long sequence of global states that begins with a global state in which all input values are a, ends with a global state in which all input values are b, and in between systematically changes input values from a to b. These changes are made so gradually, however, that for any two adjacent global states in the sequence, at most one processor can distinguish them; that is, they are *similar* to all other processors. Second, we label the points of B with global states. We label the points on each edge of B with a similarity chain of global states. For example, consider the edge between the corner where all processors start with input value a and the corner where all processors start with b. We label this edge with the similarity chain for a and b, as

described above. We label each remaining point using a combination of the global states on the edges. Third, we assign nonfaulty processors to points in such a way that the processor labeling a point has the same local state in the global states labeling all adjacent points. Finally, we project each global state onto the associated nonfaulty processor's local state, and label the point with the resulting processor-state pair.

4 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. A protocol has two components: a *message component* that maps a processor's local state to the list of messages it should send in the next round, and an *output component* that maps a processor's local state to the output value (if any) that it should choose. 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 required by the protocol 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

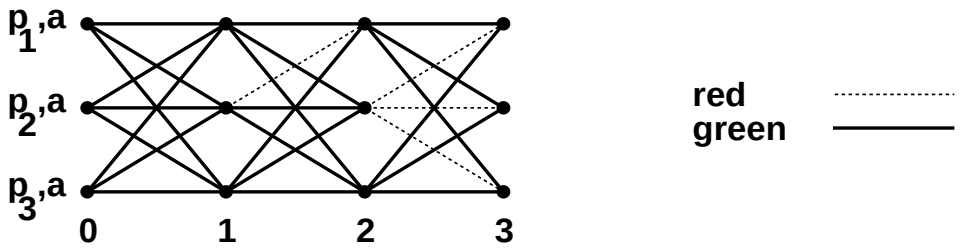

Figure 9: A three-round communication graph.

an r-round execution of P is a two-dimensional 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 9 illustrates a three round communication graph. In this figure, green edges are denoted by solid lines and red edges by dashed lines. 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 the 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 crash failure model, a processor is silent in all rounds following the round in which it stops. This means that all communication graphs representing executions in this model 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, which means that p is silent from round $i+1$. We assume that all communication graphs in this paper have this property, and we note that every r-round graph with this property corresponds to an r-round execution of P.

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 $\mathcal{G}$ con-

taining all the information visible to p at the end of round t. Namely, $\mathcal{G}(p,t)$ is the subgraph induced by the node $\langle p,t\rangle$ and all prior nodes $\langle q,t'\rangle$ such that $t' < t$ and there is a path from $\langle q,t'\rangle$ to $\langle p,t\rangle$ consisting of 0 or 1 red edges followed by 0 or more green edges.

In the remainder of this paper, we use graphs to represent states. Wherever we used "state" in the informal overview of Section 3, we now substitute the word "graph." Furthermore, we defined a full-information protocol to be a protocol in which processors broadcast their local states in every round, but we now assume that processors broadcast the local communication graphs representing these local states instead (as in [MT88]). In addition, we assume that all executions of a full-information protocol run for exactly r rounds and produce output at exactly time r. All local and global communication graphs are graphs at time r, unless otherwise specified.

The crucial property of a full-information protocol is that every protocol can be simulated by a full-information protocol, and hence that we can restrict attention to full-information protocols when proving the lower bound in this paper:

Lemma 4: If there is an n-processor protocol solving k-set agreement with f failures in r rounds, then there is an n-processor full-information protocol solving k-set agreement with f failures in r rounds.

5 Constructing the Bermuda Triangle

In this section, we define the basic geometric constructs used in our lower bound proof. We begin with some preliminary definitions. A *simplex* S is the convex hull of $k+1$ affinely-independent[1] 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 at most 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[2] for all pairs i and j. The *vertices* of a triangulation are

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

[2] 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.

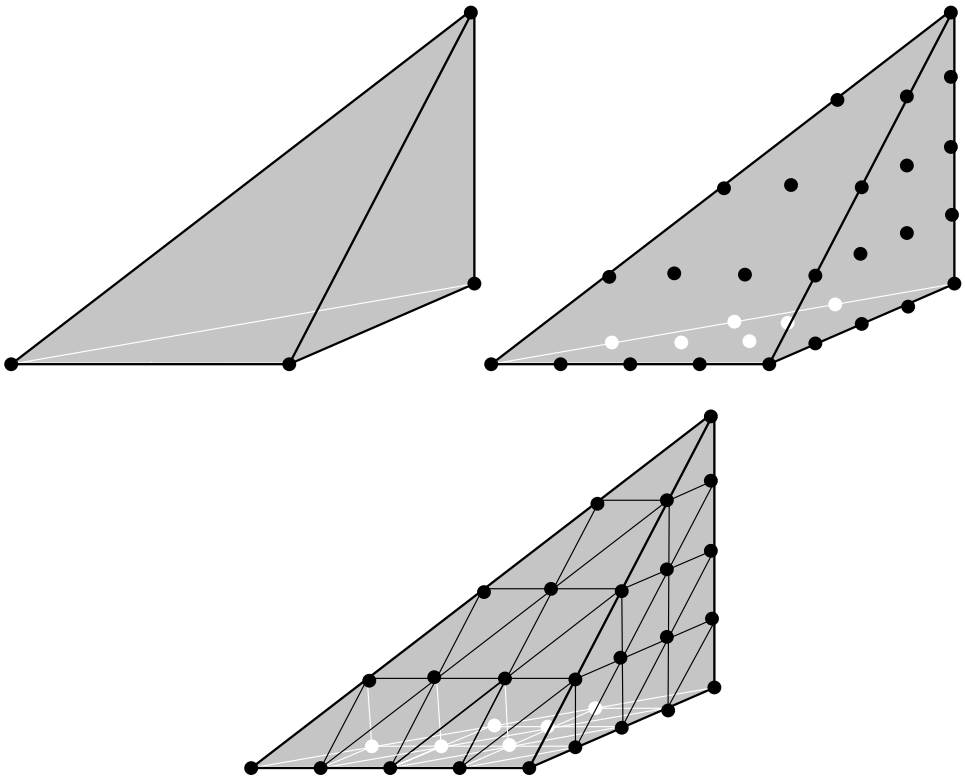

Figure 10: Construction of Bermuda Triangle.

the vertices of the S_i. Any triangulation of S induces triangulations of its faces in the obvious way.

The construction of the Bermuda Triangle is illustrated in Figure 10. 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, one of the six simplexes illustrated in Figure 11 that can be glued together to form a unit cube. The triangulation B consists of all such simplexes

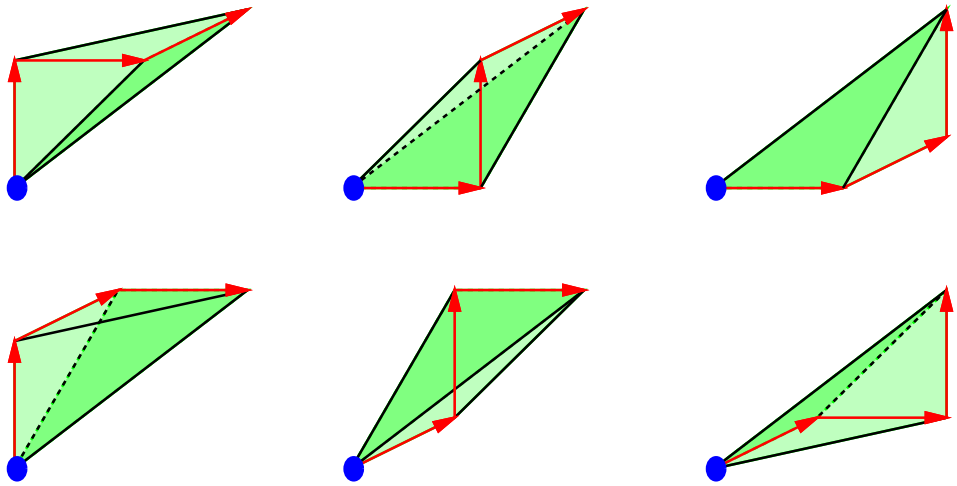

Figure 11: Simplex generation in Kuhn's triangulation.

filling the interior of B. For example, the 2-dimensional Bermuda Triangle is illustrated in Figure 7. 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 "edges" of $\mathcal{B}$ are partitioned to form the edges of B. More formally, the triangulation B induces triangulations of the one-dimensional faces (line segments connecting the vertices) 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 3, 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 processor-graph labeling is the primary technical challenge confronted in this paper, and the subject of the next three sections. We give a three-step procedure for labeling each vertex with a processor p and a

global communication graph $\mathcal{G}$, and then we simply replace $\mathcal{G}$ with p's local communication graph $\mathcal{L}$ in $\mathcal{G}$ to obtain the final labeling. In Section 6, we show how to construct long chains of global communication graphs in which adjacent graphs are nearly indistinguishable, and we use these sequences of graphs to label the vertices along the edges of the Bermuda Triangle. In Section 7, we show how to merge these graphs labeling vertices along the edges to obtain the graphs labeling the interior vertices. Finally, in Section 8, we show how to assign processors to vertices.

6 Step 1: Similarity chain construction

In this section, we show how to construct exponentially long sequences of global communication graphs in which processor input values are slowly changed from one value to another as we move along the sequence. These chains are called *similarity chains* since each pair of adjacent graphs in the sequence differ so little that the two graphs are indistinguishable (or similar) to all but perhaps one processor. Our technique for constructing these sequences is essentially that of Dwork and Moses [DM90] and Moses and Tuttle [MT88], with two important differences. First, we augment the definition of a communication graph to include tokens on nodes of the graph to represent processor failures. By shifting these tokens among the nodes as we move from graph to graph in the similarity chain, we can simplify some bookkeeping required to assign processor ids to vertices of the Bermuda Triangle in Section 8. Second, instead of defining sequences of graphs directly, we define a small collection of graph operations that make minor changes to a graph, so that adjacent graphs in the sequence are the result of applying a graph operation to the first graph to generate the second. This additional machinery helps us reason about the graphs appearing in similarity chains when we use these graphs to compute the graphs assigned to nodes on the interior of the Bermuda Triangle in Section 7. Because our operations make changes that are so small and because of the token shifting, our similarity chains are even longer and the difference between adjacent graphs are even more minute than in similarity chains used in prior work.

6.1 Augmented communication graphs

We augment the definition of a communication graph by placing tokens on the nodes of a 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 p in some round $j \leq i$ no later than round i (Figure 12). In this sense, every processor failure is "covered"

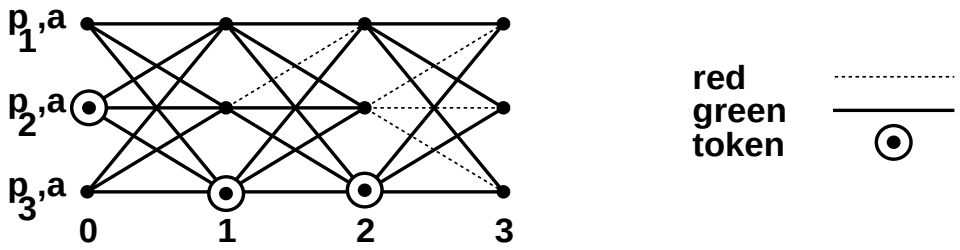

Figure 12: Three-round communication graph with one token per round.

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* $\mathcal{G}$, an *augmented communication graph*, to be a communication graph with zero or more tokens placed on each node of the graph in a way 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 that 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. In particular, when $r \leq f/k$, at most $kr \leq f$ processors fail in a k-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 (see Figure 13):

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 conditions of the form "p is covered in round i" are omitted.

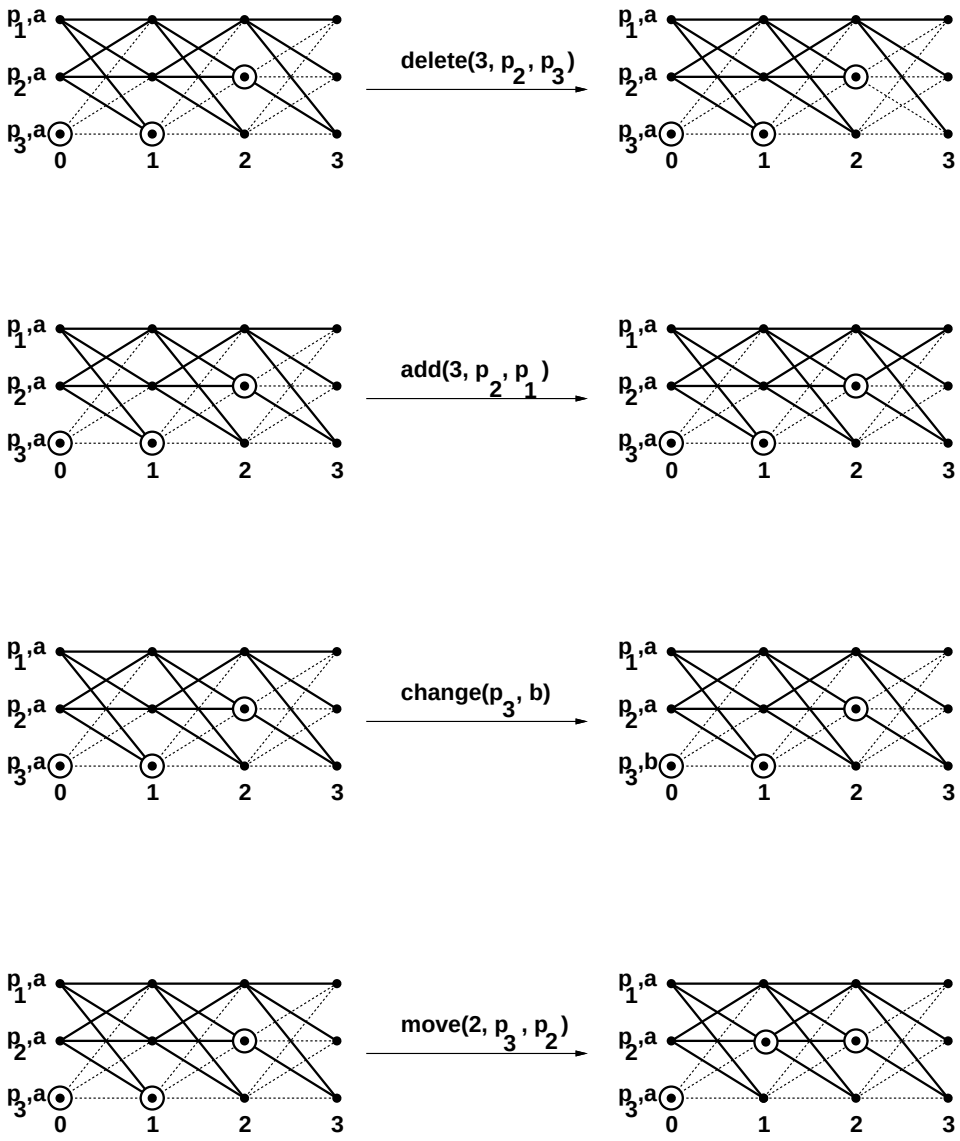

Figure 13: Operations on augmented communication graphs.

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. It is crucial to our construction that these sequences $\sigma[v]$ differ only in the value v to which the input values are being changed by the sequence. For example, we might have two sequences

$$\begin{aligned}\sigma[a] &= \cdots \mathit{delete}(1,p,p_n)\mathit{change}(p,a)\mathit{add}(1,p,p_n)\cdots \\ \sigma[b] &= \cdots \mathit{delete}(1,p,p_n)\mathit{change}(p,b)\mathit{add}(1,p,p_n)\cdots\end{aligned}$$

changing input values to a and b, respectively, but these sequences are identical except for the fact that the value a appears in one exactly where the value b appears in the other. For this reason, it is convenient to be able to replace the values a and b appearing in the two sequences with a single symbol $\textsc{v}$ and define a parameterized sequence

$$\sigma[\textsc{v}] = \cdots \mathit{delete}(1,p,p_n)\mathit{change}(p,\textsc{v})\mathit{add}(1,p,p_n)\cdots$$

so that the two sequences $\sigma[a]$ and $\sigma[b]$ can be obtained simply by replacing the symbol $\textsc{v}$ with a and b, respectively. We define a *parameterized sequence* $\sigma[\textsc{v}]$ to be a sequence of graph operations with the symbol $\textsc{v}$ appearing as a parameter to some of the graph operations in the sequence. In what follows, we will construct a parameterized sequence $\sigma[\textsc{v}]$ with the property that for all values v and all graphs $\mathcal{G}$, the sequence $\sigma[v]$ transforms $\mathcal{G}$ into $\mathcal{G}[v]$.

Given a graph $\mathcal{G}$, let $\mathit{red}(\mathcal{G},p,m)$ and $\mathit{green}(\mathcal{G},p,m)$ be graphs identical to $\mathcal{G}$ except that all edges from p in rounds $m,\ldots,r$ are red and green, respectively. We define these graphs only if

1. p is covered in round m in $\mathcal{G}$,
2. all faulty processors are silent from round m (or earlier) in $\mathcal{G}$, and
3. all tokens are on p_1 in rounds $m+1,\ldots,r$ in $\mathcal{G}$.

In addition, we define the graph $\mathit{green}(\mathcal{G},p,m)$ only if

4. p is active through round $m-1$ in $\mathcal{G}$.

If $\mathcal{G}$ is an ℓ-graph and $\mathit{red}(\mathcal{G},p,m)$ and $\mathit{green}(\mathcal{G},p,m)$ are both defined, then these conditions ensure that $\mathit{red}(\mathcal{G},p,m)$ and $\mathit{green}(\mathcal{G},p,m)$ are both ℓ-graphs.

In the case of ordinary communication graphs, a result by Moses and Tuttle [MT88] implies that there is a sequence of graphs called a "similarity chain" from $\mathcal{G}$ to $\mathit{red}(\mathcal{G}, p, m)$ and from $\mathcal{G}$ to $\mathit{green}(\mathcal{G}, p, m)$. 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. Essentially the same proof works for augmented communication graphs, provided we expand the sequence of graph operations by inserting *move* operations between the *add*, *delete*, and *change* operations to move the tokens among the nodes appropriately. With this modification, we can prove the following. Let $\mathit{faulty}(\mathcal{G})$ be the set of processors that fail in $\mathcal{G}$.

Lemma 5: For every processor p, round m, and set π of processors, there are sequences $\mathit{silence}_\pi(p, m)$ and $\mathit{revive}_\pi(p, m)$ such that for all graphs $\mathcal{G}$:

1. If $\mathit{red}(\mathcal{G}, p, m)$ is defined and $\pi = \mathit{faulty}(\mathcal{G})$, then $\mathit{silence}_\pi(p, m)(\mathcal{G}) = \mathit{red}(\mathcal{G}, p, m)$.
2. If $\mathit{green}(\mathcal{G}, p, m)$ is defined and $\pi = \mathit{faulty}(\mathcal{G})$, then $\mathit{revive}_\pi(p, m)(\mathcal{G}) = \mathit{green}(\mathcal{G}, p, m)$.

Proof: We proceed by reverse induction on m. Suppose $m = r$. Define

$$\begin{aligned} \mathit{silence}_\pi(p, r) &= \mathit{delete}(r, p, p_1) \cdots \mathit{delete}(r, p, p_n) \\ \mathit{revive}_\pi(p, r) &= \mathit{add}(r, p, p_1) \cdots \mathit{add}(r, p, p_n). \end{aligned}$$

For part 1, let $\mathcal{G}$ be any graph and suppose $\mathit{red}(\mathcal{G}, p, r)$ is defined. For each i with $0 \leq i \leq n$, let $\mathcal{G}_i$ be the graph identical to $\mathcal{G}$ except that the round r edges from p to $p_1, \ldots, p_i$ are red. Since $\mathit{red}(\mathcal{G}, p, r)$ is defined, condition 1 implies that p is covered in round r in $\mathcal{G}$. Since p is covered in round r, each $\mathcal{G}_{i-1}$ is a graph since it differs from $\mathcal{G}$ only in some new red edges from p in round r, and $\mathit{delete}(r, p, p_i)$ can be applied to $\mathcal{G}_{i-1}$ to transform it to $\mathcal{G}_i$. Since $\mathcal{G} = \mathcal{G}_0$ and $\mathcal{G}_n = \mathit{red}(\mathcal{G}, p, r)$, it follows that $\mathit{silence}_\pi(p, r)$ transforms $\mathcal{G}$ to $\mathit{red}(\mathcal{G}, p, r)$. For part 2, let $\mathcal{G}$ be any graph and suppose $\mathit{green}(\mathcal{G}, p, r)$ is defined. The proof of this part is the direct analogue of the proof of part 1. The only difference is that since we are coloring round r edges from p green instead of red, we must verify that p is active through round $r - 1$ in $\mathcal{G}$, but this follows immediately from condition 4.

Suppose $m < r$ and the induction hypothesis holds for $m+1$. Define $\pi' = \pi \cup \{p\}$ and define

$$\mathit{set}(m + 1, p_i) = \mathit{move}(m + 1, p_1, p_2) \cdots \mathit{move}(m + 1, p_{i-1}, p_i)$$

$$\mathit{reset}(m+1,p_i) \;=\; \mathit{move}(m+1,p_i,p_{i-1})\cdots\mathit{move}(m+1,p_2,p_1).$$

The *set* function moves the token from p_1 to p_i and the *reset* function moves the token back from p_i to p_1.
Define $\mathit{block}(m,p,p_i)$ to be $\mathit{delete}(m,p,p_i)$ if $p_i \in \pi'$, and otherwise

$$\mathit{silence}_{\pi'}(p_i,m+1) \quad \begin{array}{c} \mathit{set}(m+1,p_i) \\ \mathit{delete}(m,p,p_i) \\ \mathit{reset}(m+1,p_i). \end{array} \quad \mathit{revive}_{\pi'\cup\{p_i\}}(p_i,m+1)$$

Define $\mathit{unblock}(m,p,p_i)$ to be $\mathit{add}(m,p,p_i)$ if $p_i \in \pi'$, and otherwise

$$\mathit{silence}_{\pi'}(p_i,m+1) \quad \begin{array}{c} \mathit{set}(m+1,p_i) \\ \mathit{add}(m,p,p_i) \\ \mathit{reset}(m+1,p_i). \end{array} \quad \mathit{revive}_{\pi'\cup\{p_i\}}(p_i,m+1)$$

Finally, define

$$\begin{aligned} \mathit{block}(m,p) &= \mathit{block}(m,p,p_1)\cdots\mathit{block}(m,p,p_n) \\ \mathit{unblock}(m,p) &= \mathit{unblock}(m,p,p_1)\cdots\mathit{unblock}(m,p,p_n) \end{aligned}$$

and define

$$\begin{aligned} \mathit{silence}_\pi(p,m) &= \mathit{silence}_\pi(p,m+1)\,\mathit{block}(m,p) \\ \mathit{revive}_\pi(p,m) &= \mathit{silence}_\pi(p,m+1)\,\mathit{unblock}(m,p)\,\mathit{revive}_{\pi'}(p,m+1). \end{aligned}$$

For part 1, let $\mathcal{G}$ be any graph, and suppose $\mathit{red}(\mathcal{G},p,m)$ is defined and $\pi = \mathit{faulty}(\mathcal{G})$. Since $\mathit{red}(\mathcal{G},p,m)$ is defined, the graph $\mathit{red}(\mathcal{G},p,m+1)$ is also defined, and the induction hypothesis for $m+1$ states that $\mathit{silence}_\pi(p,m+1)$ transforms $\mathcal{G}$ to $\mathit{red}(\mathcal{G},p,m+1)$. We will now show that $\mathit{block}(m,p)$ transforms $\mathit{red}(\mathcal{G},p,m+1)$ to $\mathit{red}(\mathcal{G},p,m)$, and we will be done. For each i with $0 \le i \le n$, let $\mathcal{G}_i$ be the graph identical to $\mathcal{G}$ except that p is silent from round $m+1$ and the round m edges from p to $p_1,\ldots,p_i$ are red in $\mathcal{G}_i$. Since $\mathit{red}(\mathcal{G},p,m)$ is defined, condition 1 implies that p is covered in round m in $\mathcal{G}$. For each i with $0 \le i \le n$, it follows that $\mathcal{G}_i$ really is a graph and that $\pi' = \mathit{faulty}(\mathcal{G}_i)$. Since $\mathit{red}(\mathcal{G},p,m+1) = \mathcal{G}_0$ and $\mathcal{G}_n = \mathit{red}(\mathcal{G},p,m)$, it is enough to show that $\mathit{block}(m,p,p_i)$ transforms $\mathcal{G}_{i-1}$ to $\mathcal{G}_i$ for each i with $1 \le i \le n$. The proof of this fact depends on whether $p_i \in \pi'$, so we consider two cases.

Consider the easy case with $p_i \in \pi'$. We know that p is covered in round m in $\mathcal{G}_{i-1}$ since it is covered in $\mathcal{G}$ by condition 1. We know that p is silent from round $m+1$ in $\mathcal{G}_{i-1}$ since it is silent in $\mathcal{G}_0 = \mathit{red}(\mathcal{G},p,m+1)$.

We know that p_i is silent from round $m+1$ in $\mathcal{G}_{i-1}$ since $p_i \in \pi'$ implies (assuming that p_i is not just p again) that p_i fails in $\mathcal{G}$, and hence is silent from round $m+1$ in $\mathcal{G}$ by condition 2. This means that $\mathit{block}(m, p, p_i) = \mathit{delete}(m, p, p_i)$ can be applied to $\mathcal{G}_{i-1}$ to transform $\mathcal{G}_{i-1}$ to $\mathcal{G}_i$.

Now consider the difficult case when $p_i \notin \pi'$. Let $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$ be graphs identical to $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$, except that a single round $m+1$ token is on p_i in $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$. Condition 3 guarantees that all round $m+1$ tokens are on p_1 in $\mathcal{G}$, and hence in $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$, so $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$ really are graphs. In addition, $\mathit{set}(m+1, p_i)$ transforms $\mathcal{G}_{i-1}$ to $\mathcal{H}_{i-1}$, and $\mathit{reset}(m+1, p_i)$ transforms $\mathcal{H}_i$ to $\mathcal{G}_i$. Let $\mathcal{I}_{i-1}$ and $\mathcal{I}_i$ be identical to $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$ except that p_i is silent from round $m+1$ in $\mathcal{I}_{i-1}$ and $\mathcal{I}_i$. Processor p_i is covered in round $m+1$ in $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$, so $\mathcal{I}_{i-1}$ and $\mathcal{I}_i$ really are graphs. In fact, p_i does not fail in $\mathcal{G}$ since $p_i \notin \pi'$, so p_i is active through round m in $\mathcal{I}_{i-1}$ and $\mathcal{I}_i$, so $\mathcal{I}_{i-1} = \mathit{red}(\mathcal{H}_{i-1}, p_i, m+1)$ and $\mathcal{H}_i = \mathit{green}(\mathcal{I}_i, p_i, m+1)$. The inductive hypothesis for $m+1$ states that $\mathit{silence}_{\pi'}(p_i, m+1)$ transforms $\mathcal{H}_{i-1}$ to $\mathcal{I}_{i-1}$, and $\mathit{revive}_{\pi' \cup \{p_i\}}(p_i, m+1)$ transforms $\mathcal{I}_i$ to $\mathcal{H}_i$. Finally, notice that the only difference between $\mathcal{I}_{i-1}$ and $\mathcal{I}_i$ is the color of the round m edge from p to p_i. Since p is covered in round m and p and p_i are silent from round $m+1$ in both graphs, we know that $\mathit{delete}(m, p, p_i)$ transforms $\mathcal{I}_{i-1}$ to $\mathcal{I}_i$. It follows that $\mathit{block}(m, p, p_i)$ transforms $\mathcal{G}_{i-1}$ to $\mathcal{G}_i$, and we are done.

For part 2, let $\mathcal{G}$ be any graph and suppose $\mathit{green}(\mathcal{G}, p, m)$ is defined and $\pi = \mathit{faulty}(\mathcal{G})$. Since $\mathit{green}(\mathcal{G}, p, m)$ is defined, let $\mathcal{G}' = \mathit{green}(\mathcal{G}, p, m)$. Now let $\mathcal{H}$ and $\mathcal{H}'$ be graphs identical to $\mathcal{G}$ and $\mathcal{G}'$ except that p is silent from round $m+1$ in $\mathcal{H}$ and $\mathcal{H}'$. Since $\mathit{green}(\mathcal{G}, p, m)$ is defined, processor p is covered in round m in $\mathcal{G}$ by condition 1 and hence in $\mathcal{G}'$, so $\mathcal{H}$ and $\mathcal{H}'$ really are graphs. In addition, since $\mathit{green}(\mathcal{G}, p, m)$ is defined, processor p is active through round $m-1$ in $\mathcal{G}$ by condition 4, so processor p is active through round m in $\mathcal{G}'$ and $\mathcal{H}'$. This means that $\mathit{green}(\mathcal{H}', p, m+1)$ is defined, and in fact we have $\mathcal{H} = \mathit{red}(\mathcal{G}, p, m+1)$ and $\mathcal{G}' = \mathit{green}(\mathcal{H}', p, m+1)$. The induction hypothesis for $m+1$ states that $\mathit{silence}_{\pi}(p, m+1)$ transforms $\mathcal{G}$ to $\mathcal{H}$ and that $\mathit{revive}_{\pi'}(p, m+1)$ transforms $\mathcal{H}'$ to $\mathcal{G}'$. To complete the proof, we need only show that $\mathit{unblock}(m, p)$ transforms $\mathcal{H}$ to $\mathcal{H}'$. The proof of this fact is the direct analogue of the proof in part 1 that $\mathit{block}(m, p)$ transforms $\mathit{red}(\mathcal{G}, p, m+1)$ to $\mathit{red}(\mathcal{G}, p, m)$. The only difference is that since we are coloring round m edges from p with green instead of red, we must verify that p is active through round $m-1$ in the graphs $\mathcal{H}_i$ analogous to $\mathcal{G}_i$ in the proof of part 1, but this follows immediately from condition 4. □

Given a graph $\mathcal{G}$, let $\mathcal{G}_i[v]$ be a graph identical to $\mathcal{G}$, except that processor p_i has input v. Using the preceding result, we can transform $\mathcal{G}$ to $\mathcal{G}_i[v]$.

Lemma 6: 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 $\mathcal{G}_i[v]$.

Proof: Define

$$\begin{aligned} \mathit{set}(1, p_i) &= \mathit{move}(1, p_1, p_2) \cdots \mathit{move}(1, p_{i-1}, p_i) \\ \mathit{reset}(1, p_i) &= \mathit{move}(1, p_i, p_{i-1}) \cdots \mathit{move}(1, p_2, p_1) \end{aligned}$$

and define

$$\sigma_i[\mathrm{v}] = \mathit{set}(1, p_i)\mathit{silence}_{\emptyset}(p_i, 1)\mathit{change}(p_i, \mathrm{v})\mathit{revive}_{\{p_i\}}(p_i, 1)\mathit{reset}(1, p_i)$$

where $\emptyset$ denotes the empty set. Now consider any value v and any failure-free graph $\mathcal{G}$, and let $\mathcal{G}' = \mathcal{G}_i[v]$. Since $\mathcal{G}$ and $\mathcal{G}'$ are failure-free graphs, all round 1 tokens are on p_1, so let $\mathcal{H}$ and $\mathcal{H}'$ be graphs identical to $\mathcal{G}$ and $\mathcal{G}'$ except that a single round 1 token is on p_i in $\mathcal{H}$ and $\mathcal{H}'$. We know that $\mathcal{H}$ and $\mathcal{H}'$ are graphs, and that $\mathit{set}(1, p_i)$ transforms $\mathcal{G}$ to $\mathcal{H}$ and $\mathit{reset}(1, p_i)$ transforms $\mathcal{H}'$ to $\mathcal{G}'$. Since p_i is covered in $\mathcal{H}$ and $\mathcal{H}'$, let $\mathcal{I}$ and $\mathcal{I}'$ be identical to $\mathcal{H}$ and $\mathcal{H}'$ except that p_i is silent from round 1. We know that $\mathcal{I}$ and $\mathcal{I}'$ are graphs, and it follows by Lemma 5 that $\mathit{silence}_{\emptyset}(p_i, 1)$ transforms $\mathcal{H}$ to $\mathcal{I}$ and that $\mathit{revive}_{\{p_i\}}(p_i, 1)$ transforms $\mathcal{I}'$ to $\mathcal{H}'$. Finally, notice that $\mathcal{I}$ and $\mathcal{I}'$ differ only in the input value for p_i. Since p_i is covered and silent from round 1 in both graphs, the operation $\mathit{change}(p_i, v)$ can be applied to $\mathcal{I}$ and transforms it to $\mathcal{I}'$. Concatenating these transformations, it follows that $\sigma_i[v]$ transforms $\mathcal{G}$ to $\mathcal{G}' = \mathcal{G}_i[v]$. □

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

Lemma 7: Let $\sigma[\mathrm{v}] = \sigma_1[\mathrm{v}] \cdots \sigma_n[\mathrm{v}]$. For every value v and 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.

7 Step 2: Graph assignment

In this section, we label each vertex of B with an augmented global communication graph. Speaking informally, we use each sequence $\sigma[v_i]$ of graph operations to generate a sequence of graphs, and then use this sequence of graphs to label the vertices along the edge of the Bermuda Triangle in the ith dimension. To label the vertices on the interior of the Bermuda Triangle, we define an operation to "merge" graphs labeling the edges into a single graph.

7.1 Graph merge

We begin with the definition of the operation merging a sequence of graphs into a single graph.

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, let $x = (x_1, \ldots, x_k)$ be a vertex of B and let $\mathcal{H}_i$ be the result of applying the first x_i operations in $\sigma[v_i]$ to some fixed graph $\mathcal{G}$. It follows from the construction of $\sigma[\mathrm{V}]$ that there is a single integer c with the property that it is the cth graph operation in $\sigma[v_i]$ that changes p's input to v_i for each i. Since $x = (x_1, \ldots, x_k)$ is a vertex of B, the indices x_i are ordered by $x_1 \geq x_2 \geq \cdots \geq x_k$. It follows that p's input has been changed to v_i in $\mathcal{H}_i$ for those $i = 1, \ldots, j$ satisfying $x_i \geq c$ and not in $\mathcal{H}_i$ for those $i = j+1, \ldots, k$ satisfying $c > x_i$. The second condition above is just a compact way of identifying this final value v_j.

One important property of the merge operator is that merging a sequence of 1-graphs yields a k-graph, where k is the length of the sequence of 1-graphs:

Lemma 8: 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.

Proof: We consider the three conditions required of a k-graph in turn. First, there are k tokens in each round of $\mathcal{H}$ since there is 1 token in each round of each graph $\mathcal{H}_1, \ldots, \mathcal{H}_k$. Second, every red edge in $\mathcal{H}$ is covered by a token since every red edge in $\mathcal{H}$ corresponds to a red edge in one of the graphs $\mathcal{H}_j$, and this edge is covered by a token in $\mathcal{H}_j$. Third, if there is a red edge from p in round i in $\mathcal{H}$, then there is a red edge from p in round i

of one of the graphs $\mathcal{H}_j$. In this graph, p is silent from round $i+1$, so the same is true in $\mathcal{H}$. Thus, $\mathcal{H}$ is a k-graph. □

Remember that at most $kr \leq f$ processors fail in a k-graph like $\mathcal{H}$.

7.2 Graph assignment

Now we can define the assignment of graphs to vertices of B. For each input 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 8, 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.

7.3 Graph consistency

The graphs labeling the vertices of a primitive simplex have some convenient properties. For this section, fix a primitive simplex S, 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 9: 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.

Proof: Suppose to the contrary that q is faulty in some graph labeling a vertex of S. Let $y_0 = (a_1, \ldots, a_k)$ be the first vertex of S. For each i, let σ_i and $\sigma_i\tau_i$ be the prefixes of $\sigma[v_i]$ consisting of the first a_i and $a_i + 1$ operations, and let $\mathcal{H}_i$ and $\mathcal{H}'_i$ be the result of applying σ_i and $\sigma_i\tau_i$ to $\mathcal{F}_{i-1}$. For each i, we know that the graph $\mathcal{G}_i$ labeling the vertex y_i of S is the merge of graphs $\mathcal{I}^i_1, \ldots, \mathcal{I}^i_k$ where $\mathcal{I}^i_j$ is either $\mathcal{H}_j$ or $\mathcal{H}'_j$. Suppose q is faulty in $\mathcal{G}_i$. Then q must be faulty in some graph $\mathcal{I}^i_j$ in the sequence of graphs $\mathcal{I}^i_1, \ldots, \mathcal{I}^i_n$ merged to form $\mathcal{G}_i$, so q must fail in one of the graphs $\mathcal{H}_j$ or $\mathcal{H}'_j$. Since σ_j and $\sigma_j\tau_j$ are prefixes of $\sigma[v_j]$, it is easy to see from the definition of $\sigma[v_j]$ that

the fact that q fails in one of the graphs $\mathcal{H}_j$ and $\mathcal{H}'_j$ implies that q is covered in both graphs. Since one of these graphs is contained in the sequence of graphs merged to form $\mathcal{G}_a$ for each a, it follows that q is covered in each $\mathcal{G}_a$. This contradicts the fact that q is uncovered in a graph labeling a 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 10: If F_i is the set of processors failing in $\mathcal{G}_i$ and $F = \cup_i F_i$, then $|F| \leq rk \leq f$.

Proof: If $q \in F$, then $q \in F_i$ for some i and q fails in $\mathcal{G}_i$, so q is covered in every graph labeling every vertex of S by Lemma 9. It follows that each processor in F is covered in each graph labeling S. Since there are at most rk tokens to cover processors in any graph, there are at most rk processors in 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 11: 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 q to p in round t, then p and q are silent from round $t+1$ in $\mathcal{G}_0, \ldots, \mathcal{G}_k$.

Proof: Suppose the two graphs $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$ labeling vertices y_{i-1} and y_i differ in the input to p at time $t = 0$ or in the color of an edge from q to p in round t. The vertices differ in exactly one coordinate j, so $y_{i-1} = (a_1, \ldots, a_j, \ldots, a_k)$ and $y_i = (a_1, \ldots, a_j + 1, \ldots, a_k)$. For each ℓ, let σ_ℓ be the prefix of $\sigma[v_\ell]$ consisting of the first a_ℓ operations, and let $\mathcal{H}^0_\ell$ be the result of applying σ_ℓ to $\mathcal{F}_{\ell-1}$. Furthermore, in the special case of $\ell = j$, let $\sigma_j \tau_j$ be the prefix of $\sigma[v_j]$ consisting of the first $a_j + 1$ operations, and let $\mathcal{H}^1_j$ be the result of applying $\sigma_j \tau_j$ to $\mathcal{F}_{j-1}$.

We know that $\mathcal{G}_{i-1}$ is the merge of $\mathcal{H}^0_1, \ldots, \mathcal{H}^0_j, \ldots, \mathcal{H}^0_k$, and that $\mathcal{G}_i$ is the merge of $\mathcal{H}^0_1, \ldots, \mathcal{H}^1_j, \ldots, \mathcal{H}^0_k$. If $\mathcal{H}^0_j$ and $\mathcal{H}^1_j$ are equal, then $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$

are equal. Thus, $\mathcal{H}_j^0$ and $\mathcal{H}_j^1$ must differ in the input to p at time $t = 0$ or in the color of an edge between q and p in round t, exactly as $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$ differ. Since $\mathcal{H}_j^0$ and $\mathcal{H}_j^1$ are the result of applying σ_j and $\sigma_j\tau_j$ to $\mathcal{F}_{j-1}$, this change at time t must be caused by the operation τ_j. It is easy to see from the definition a graph operation like τ_j that (1) if τ_j changes p's input value, then p is silent from round 1 in $\mathcal{H}_j^0$ and $\mathcal{H}_j^1$, and (2) if τ_j changes the color of an edge from q to p in round t, then p and q are silent from round $t+1$ in $\mathcal{H}_j^0$ and $\mathcal{H}_j^1$. Consequently, the same is true in the merged graphs $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$. □

Lemma 12: 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 $\mathcal{G}_0, \ldots, \mathcal{G}_k$.

Proof: We proceed by induction on t. If $t = 0$, then the two graphs must differ in the input to p at time 0, and Lemma 11 implies that p is silent from round 1 in the graphs $\mathcal{G}_0, \ldots, \mathcal{G}_k$ labeling the simplex. Suppose $t > 0$ and the inductive hypothesis holds for $t-1$. Processor p's local communication graph at time t can differ in the two graphs for one of two reasons: either p hears from some processor q in round t in one graph and not in the other, or p hears from some processor q in both graphs but q has different local communication graphs at time $t-1$ in the two graphs. In the first case, Lemma 11 implies that p is silent from round $t+1$ in the graphs $\mathcal{G}_0, \ldots, \mathcal{G}_k$. In the second case, the induction hypothesis for $t-1$ implies that q is silent from round t in the graphs $\mathcal{G}_0, \ldots, \mathcal{G}_k$. In particular, q is silent in round t in $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$, so it is not possible for p to hear from q in round t in both graphs, and this case is impossible. □

Lemma 13: 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 $\mathcal{G}_0, \ldots, \mathcal{G}_k$.

Proof: If p has different local communication graphs at time $r-1$ in two of the graphs $\mathcal{G}_0, \ldots, \mathcal{G}_k$, then there are two adjacent graphs $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$ in which p has different local communication graphs at time $r-1$. By Lemma 12, p is silent in round r in all of the graphs $\mathcal{G}_0, \ldots, \mathcal{G}_k$, contradicting the hypothesis that p sent a round r message in one of them. □

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

Lemma 14: Given a local processor labeling, 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 $\mathcal{G}$.

Proof: Let Q be the set of processors that send a round r message in any of the graphs $\mathcal{G}_0, \ldots, \mathcal{G}_k$. Notice that this set includes the uncovered processors $q_0, \ldots, q_k$, since Lemma 9 says that these processors are nonfaulty in each of these graphs. For each processor $q \in Q$, Lemma 13 says that q has the same local communication graph at time $r-1$ in each graph $\mathcal{G}_0, \ldots, \mathcal{G}_k$.

Let $\mathcal{H}$ be the global communication graph underlying any one of these graphs. Notice that each processor $q \in Q$ is active through round $r-1$ in $\mathcal{H}$. To see this, notice that since q sends a message in round r in one of the graphs labeling S, it sends all messages in round $r-1$ in that graph. On the other hand, if q fails to send a message in round $r-1$ in $\mathcal{H}$, then the same is true for the corresponding graph labeling S. Thus, there are adjacent graphs $\mathcal{G}_{i-1}$ and $\mathcal{G}_i$ labeling S where p sends a round $r-1$ message in one and not in the other. Consequently, Lemma 11 says q is silent in round r in all graphs labeling S, but this contradicts the fact that q does send a round r message in one of these graphs.

Now let $\mathcal{G}$ be the global communication graph obtained from $\mathcal{H}$ by coloring green each round r edge from each processor $q \in Q$, unless the edge is red in one of the local communication graphs $\mathcal{L}_0, \ldots, \mathcal{L}_k$ in which case we color it red in $\mathcal{G}$ as well. Notice that since the processors $q \in Q$ are active through round $r-1$ in $\mathcal{H}$, changing the color of a round r edge from a processor $q \in Q$ to either red or green is acceptable, provided we do not cause more that f processors to fail in the process. Fortunately, Lemma 10 implies that there are at least $n - rk \geq n - f$ processors that do not fail in any of the graphs $\mathcal{G}_0, \ldots, \mathcal{G}_k$. This means that there is a set of $n-f$ processors that send to every processor in round r of every graph $\mathcal{G}_i$, and in particular that the round r edges from these processors are green in every local communication graph $\mathcal{L}_i$. It follows that for at least $n-f$ processors, all round r edges from these processors are green in $\mathcal{G}$, so at most f processors fail in $\mathcal{G}$.

Each processor q_i is nonfaulty in $\mathcal{G}$, since q_i is nonfaulty in each $\mathcal{G}_0, \ldots, \mathcal{G}_k$, meaning each edge from q_i is green in each $\mathcal{G}_0, \ldots, \mathcal{G}_k$ and $\mathcal{L}_0, \ldots, \mathcal{L}_k$, and therefore in $\mathcal{G}$. In addition, each processor q_i has the local communication graph $\mathcal{L}_i$ in $\mathcal{G}$. To see this, notice that $\mathcal{L}_i$ consists of a round r edge from p_j to q_i for each j, and the local communication graph for p_j at time $r-1$ if this edge is green. This edge is green in $\mathcal{L}_i$ if and only if it is green in $\mathcal{G}$.

In addition, if this edge is green in $\mathcal{L}_i$, then it is green in $\mathcal{G}_i$. In this case, Lemma 13 says that p_j has the same local communication graph at time $r-1$ in each graph $\mathcal{G}_0, \ldots, \mathcal{G}_k$, and therefore in $\mathcal{G}$. Consequently, q_i has the local communication graph $\mathcal{L}_i$ in $\mathcal{G}$. □

8 Step 3: Processor assignment

What Lemma 14 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 each vertex of B.

8.1 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 $j \leq i$.

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? If we choose a low processor like p_1, then we have changed the rank of a low processor like p_2 from 2 to 1. If we choose a high processor like p_n, then we have changed the rank of a high processor like p_{n-1} from $n-3$ to $n-2$. On the other hand, if we choose to remove p_i again, then no processors change

```
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 14: The construction of *live*($\mathcal{G}$).

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 given in Figure 14 computes *live*($\mathcal{G}$). In this algorithm, processor p_i is denoted by its index i. We refer to the ith iteration of the main loop as the ith step of the construction. This construction has two obvious properties:

Lemma 15: If $i \in$ *live*($\mathcal{G}$) then

1. *i is uncovered*: $m_i = 0$
2. *room exists under i*: $\sum_{j=1}^{i-1} m_j \leq i - 1$

Proof: Suppose $i \in$ *live*($\mathcal{G}$). For part 1, if $m_i > 0$ then i will be removed by step i if it has not already removed by an earlier step, contradicting $i \in$ *live*($\mathcal{G}$). For part 2, notice that steps 1 through $i - 1$ remove a total of $\sum_{j=1}^{i-1} m_j$ values. If this sum is greater than $i - 1$, then it is not possible for all of these values to be contained in $1, \ldots, i - 1$, so i will be removed within the first $i - 1$ steps, contradicting $i \in$ *live*($\mathcal{G}$). □

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

Proof: If p is covered in $\mathcal{G}_\ell$ for some ℓ between i and j, then p is uncovered in $\mathcal{G}_{\ell-1}$ and covered in $\mathcal{G}_\ell$ for some ℓ between i and j. Since $\mathcal{G}_{\ell-1}$ and $\mathcal{G}_\ell$ are on adjacent vertices of the simplex, the sequences of graphs merged to construct them are of the form $\mathcal{H}_1, \ldots, \mathcal{H}_m, \ldots, \mathcal{H}_k$ and $\mathcal{H}_1, \ldots, \mathcal{H}'_m, \ldots, \mathcal{H}_k$, respectively, for some m. Since p is uncovered in $\mathcal{G}_{\ell-1}$ and covered in $\mathcal{G}_\ell$, it must be that p is uncovered in $\mathcal{H}_m$ and covered in $\mathcal{H}'_m$. Notice, however, that $\mathcal{H}'_m$ is used in the construction of each graph $\mathcal{G}_\ell, \mathcal{G}_{\ell+1}, \ldots, \mathcal{G}_j$. This means that p is covered in each of these graphs, contradicting the fact that p is uncovered in $\mathcal{G}_j$. □

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 17: 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 $\mathit{live}(\mathcal{G}_j)$.

Proof: Assume without loss of generality that $i < j$. Since $p \in \mathit{live}(\mathcal{G}_i)$ and $p \in \mathit{live}(\mathcal{G}_j)$, Lemma 15 implies that p is uncovered in the graphs $\mathcal{G}_i$ and $\mathcal{G}_j$, and Lemma 16 implies that p is uncovered in $\mathcal{G}_i, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_j$. Since token placements in adjacent graphs differ in at most the movement of one token from one processor to an adjacent processor, and since p is uncovered in all of these graphs, this means that the number of tokens on processors smaller than p is the same in all of these graphs. Specifically, the sum $\sum_{\ell=1}^{p-1} m_\ell$ of multiplicities of processors smaller than p is the same in $\mathcal{G}_i, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_j$. In particular, Lemma 15 implies that this sum is the same value $s \leq p-1$ in $\mathcal{G}_i$ and $\mathcal{G}_j$, so p has the same rank $p - s$ in $\mathit{live}(\mathcal{G}_i)$ and $\mathit{live}(\mathcal{G}_j)$. □

8.2 Processor assignment

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 18: $\mathit{plane}(x) \neq \mathit{plane}(y)$ if x and y are distinct vertices of the same simplex.

Proof: Since x and y are in the same simplex, we can write $y = x + f_1 + \cdots + f_j$ for some distinct unit vectors $f_1, \ldots, f_j$ and some $1 \leq j \leq k$. If $x = (x_1, \ldots, x_k)$ and $y = (y_1, \ldots, y_k)$, then the sums $\sum_{i=1}^{k} x_i$ and $\sum_{i=1}^{k} y_i$ differ by exactly j. Since $1 \leq j \leq k$ and since planes are defined as sums modulo $k+1$, we have $\mathit{plane}(x) \neq \mathit{plane}(y)$. □

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 19: The mapping π is a global processor labeling.

Proof: First, it is clear that π maps each vertex x labeled with a graph $\mathcal{G}_x$ to a processor q_x that is uncovered in $\mathcal{G}_x$. Second, π maps distinct vertices of a simplex to distinct processors. To see this, suppose to the contrary that both x and y are labeled with p, and let $\mathcal{G}_x$ and $\mathcal{G}_y$ be the graphs labeling x and y. We know that the rank of p in $\mathit{live}(\mathcal{G}_x)$ is $\mathit{plane}(x)$ and that the rank of p in $\mathit{live}(\mathcal{G}_y)$ is $\mathit{plane}(y)$, and we know that p has the same rank in $\mathit{live}(\mathcal{G}_x)$ and $\mathit{live}(\mathcal{G}_y)$ by Lemma 17. Consequently, $\mathit{plane}(x) = \mathit{plane}(y)$, contradicting Lemma 18. □

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

Now that we have assigned a global communication graph $\mathcal{G}$ and a processor p to each vertex x of the Bermuda Triangle, let us replace the pair $(p, \mathcal{G})$ labeling x with the pair $(p, \mathcal{L})$ where $\mathcal{L}$ is processor p's local communication graph in $\mathcal{G}$. The following result is a direct consequence of Lemmas 14 and 19. It says that the local communication graphs of processors labeling the corners of a simplex are consistent with a single global communication graph.

Lemma 20: 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 Finishing the proof with 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.

Notice that the corners of B are points c_i of the form $(N, \ldots, N, 0, \ldots, 0)$ with i indices of value N for $0 \leq i \leq k$. For example, $c_0 = (0, \ldots, 0)$, $c_1 = (N, 0, \ldots, 0)$, and $c_k = (N, \ldots, N)$. 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 21 (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. We will now expand the argument sketched in the introduction to show that χ_P is a Sperner coloring.

We first prove a simple claim. Recall that $\mathcal{B}$ is the simplex whose vertices are the corner vertices $c_0, \ldots, c_k$, and that B is a triangulation of $\mathcal{B}$. Let $\mathcal{F}$ be some face of $\mathcal{B}$ *not* containing the corner c_i, and let F denote the triangulation of $\mathcal{F}$ induced by B. We prove the following technical statement about vertices in F.

Claim 22: If $x = (x_1, \ldots, x_k)$ is a vertex of a face F not containing c_i, then

1. if $i = 0$, then $x_1 = N$,
2. if $0 < i < k$, then $x_{i+1} = x_i$, and
3. if $i = k$, then $x_k = 0$.

Proof: Each vertex x of B can be expressed using *barycentric coordinates* with respect to the corner vertices: that is, $x = \alpha_0 c_0 + \cdots + \alpha_k c_k$, where $0 \leq$

$\alpha_j \leq 1$ for $0 \leq j \leq k$ and $\sum_{i=0}^{k} \alpha_i = 1$. Since x is a vertex of a face F not containing the corner c_i, it follows that $\alpha_i = 0$. We consider the three cases.
Case 1: $i = 0$. Each corner $c_1, \ldots, c_k$ has the value N in the first position. Since $\alpha_0 = 0$, the value in the first position of $\alpha_0 c_0 + \cdots + \alpha_k c_k$ is $(\alpha_1 + \cdots + \alpha_k)N = N$.
Case 2: $0 < i < k$. Each corner $c_0, \ldots, c_{i-1}$ has 0 in positions i and $i+1$, and each corner $c_{i+1}, \ldots, c_k$ has N in positions i and $i+1$. Since $\alpha_i = 0$, the linear combination $\alpha_0 c_0 + \cdots + \alpha_k c_k$ will have the same value $(\alpha_{i+1} + \cdots + \alpha_k)N$ in positions i and $i+1$. Thus, $x_i = x_{i+1}$.
Case 3: $i = k$. Each corner $c_0, \ldots, c_{k-1}$ has 0 in position k. Since $\alpha_k = 0$, the value in the kth position of $\alpha_0 c_0 + \cdots + \alpha_k c_k$ is 0. Thus, $x_k = 0$. □

Lemma 23: 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.

Proof: We must show that χ_P satisfies the two conditions of a Sperner coloring.

For the first condition, consider any corner vertex c_i. Remember that c_i was originally labeled with the 1-graph $\mathcal{F}_i$ describing a failure-free execution in which all processors start with input v_i, and that the local communication graph $\mathcal{L}$ labeling c_i is a subgraph of $\mathcal{F}_i$. Since the validity condition of the k-set agreement problem requires that any value chosen by a processor must be an input value of some processor, all processors must chose v_i in $\mathcal{F}_i$, and it follows that the vertex c_i must be colored with v_i. This means that each corner c_i is colored with a distinct value v_i.

For the second condition, consider any face F of B, and let us prove that vertices in F are colored with the colors on the corners of F. Equivalently, suppose that c_i is not a corner of F, and let us prove that no vertex in F is colored with v_i.

Consider the global communication graph $\mathcal{G}$ originally labeling a vertex x of F, and the graphs $\mathcal{H}_1, \ldots, \mathcal{H}_k$ used in the merge defining $\mathcal{G}$. The definition of this merge says that the input value labeling a node $\langle p, 0 \rangle$ in $\mathcal{G}$ is v_m where m is the maximum m such that $\langle p, 0 \rangle$ is labeled with v_m in $\mathcal{H}_m$, or v_0 if no such m exists. Again, we consider three cases. In each case, we show that no processor in $\mathcal{G}$ has the input value v_i.

Suppose $i = 0$. Since $x_1 = N$ by Claim 22, we know that $\mathcal{H}_1 = \mathcal{F}_1$, where the input value of every processor is v_1. By the definition of the merge operation, it follows immediately that no processor in $\mathcal{G}$ can have input value v_0.

Suppose $1 < i < k$. Again, $x_{i+1} = x_i$ by Claim 22. Now, $\mathcal{H}_i$ is the result of applying σ_i, the first x_i operations of $\sigma[v_i]$, to the graph $\mathcal{F}_{i-1}$. Similarly, $\mathcal{H}_{i+1}$ is the result of applying σ_{i+1}, the first x_{i+1} operations of $\sigma[v_{i+1}]$, to the graph $\mathcal{F}_i$. Since $x_{i+1} = x_i$, both σ_i and σ_{i+1} are of the same length, and it follows that σ_i contains an operation of the form $change(p, v_i)$ if and only if σ_{i+1} contains an operation of the form $change(p, v_{i+1})$. This implies that for any processor, either its input value is v_{i-1} in $\mathcal{H}_i$ and v_i in $\mathcal{H}_{i+1}$, or its input value is v_i in $\mathcal{H}_i$ and v_{i+1} in $\mathcal{H}_{i+1}$. In both cases, v_i is not the input value of this processor.

Suppose $i = k$. Since $x_k = 0$ by Claim 22, we know that $\mathcal{H}_k = \mathcal{F}_{k-1}$, where the input value of every processor is v_{k-1}. By the definition of *merge*, it follows immediately that no processor in $\mathcal{G}$ can have input value v_k.

Therefore, we have shown that if x is a vertex of a face F of B, and c_i is not a corner vertex of F, then the communication graph $\mathcal{G}$ corresponding to x contains no processor with input value v_i. Therefore, by the validity condition, the value chosen at this vertex cannot be v_i, and it follows that x is assigned a color other than v_i. So, x must be colored by a color v_j such that c_j is a corner vertex of F. Since c_j is colored v_j, the second condition of Sperner's Lemma holds. So χ_P is a Sperner coloring. □

Sperner's Lemma guarantees that some primitive simplex is colored by $k+1$ distinct values, and this simplex corresponds to a global state in which $k+1$ processors choose $k+1$ distinct values, contradicting the definition of k-set agreement:

Theorem 24: If $n \geq f + k + 1$, then no protocol for k-set agreement can halt in fewer than $\lfloor f/k \rfloor + 1$ rounds.

Proof: Suppose P is a protocol for k-set agreement tolerating f faults and halting in $r \leq \lfloor f/k \rfloor$ rounds, and consider the corresponding Bermuda Triangle B. Lemma 23 says that χ_P is a Sperner coloring of B, so Sperner's Lemma 21 says that there is a simplex S whose vertices are colored with $k+1$ distinct values $v_0, \ldots, v_k$. Let $q_0, \ldots, q_k$ and $\mathcal{L}_0, \ldots, \mathcal{L}_k$ be the processors and local communication graphs labeling the corners of S. By Lemma 20, there exists a communication graph $\mathcal{G}$ in which q_i is nonfaulty and has local communication graph $\mathcal{L}_i$. This means that $\mathcal{G}$ is a time r global communication graph of P in which each q_i must choose the value v_i. In other words, $k+1$ processors must choose $k+1$ distinct values, contradicting the fact that P solves k-set agreement in r rounds. □

Acknowledgments

We are grateful to two anonymous referees for a number of suggestions leading to significant improvements in this paper. The results in the paper have appeared earlier in preliminary form. The lower bound result has appeared in [CHLT93] and the algorithm has appeared in [Cha91]. Much of 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. In addition, the first author was supported in part by NSF grant CCR-93-08103.

References

[AR96] Hagit Attiya and Sergio Rajsbaum. The combinatorial structure of wait-free solvable tasks. In *Proceedings of the 10th International Workshop on Distributed Algorithms*, volume 1151 of *Lecture Notes in Computer Science*, pages 322–343. Springer-Verlag, Berlin, October 1996.

[BG93] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 91–100, May 1993.

[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*, December 1991.

[Cha93] Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, July 1993.

[CHLT93] Soma Chaudhuri, Maurice Herlihy, Nancy Lynch, and Mark R. Tuttle. A tight lower bound for k-set agreement. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, pages 206–215, November 1993.

[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.

[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.

[GK99] Eli Gafni and Elias Koutsoupias. Three-processor tasks are undecidable. *SIAM Journal on Computing*, 28(3):970–983, 1999.

[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.

[HR94] Maurice Herlihy and Sergio Rajsbaum. Set consensus using arbitrary objects. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, pages 324–333, August 1994.

[HR95] Maurice Herlihy and Sergio Rajsbaum. Algebraic spans. In *Proceedings of the 14th Annual ACM Symposium on Principles of*

Distributed Computing, pages 90–99, August 1995. *Mathematical Structures in Computer Science*, to appear.

[HS99] Maurice P. Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, November 1999.

[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.

[SZ93] Michael Saks and Fotis Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 101–110, May 1993. *SIAM Journal on Computing*, to appear.

[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.

Contents

1 Introduction **1**

2 An optimal protocol for k-set agreement **4**

3 An overview of the lower bound proof **7**

4 The model **15**

5 Constructing the Bermuda Triangle **17**

6 Step 1: Similarity chain construction **20**
6.1 Augmented communication graphs 20
6.2 Graph operations 22
6.3 Graph sequences 24

7 Step 2: Graph assignment **28**
7.1 Graph merge 29
7.2 Graph assignment 30
7.3 Graph consistency 30

8 Step 3: Processor assignment **34**
8.1 Live processors 34
8.2 Processor assignment 36

9 Finishing the proof with Sperner's Lemma **37**