A Bound on the Rounds to Reach Lattice Agreement

Marios Mavronicolas [a,1]

[a] *Department of Computer Science, University of Cyprus, Nicosia CY-1678, Cyprus*

Abstract

The *lattice agreement* decision problem is studied in the synchronous message-passing model of distributed computation, subject to crash failures. Processors $p_1, p_2, \ldots, p_n$ start with input values $X_1, X_2, \ldots, X_n$, respectively, drawn from a *lattice* $\mathcal{L}$; the size of a maximal *chain* of elements of $\mathcal{L}$ that can be defined, starting with $\{X_1, X_2, \ldots, X_n\}$, as the *joins* of other elements is denoted *joinheight*$(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$. Each non-faulty processor chooses a value greater than or equal to its original value, and less than or equal to the join of the original values; moreover, the chosen values must be pairwise comparable. Thus, lattice agreement is a weakening of traditional *consensus*.

Early-stopping algorithms for the stronger consensus problem are known to require $\Theta(f)$ rounds of communication for any execution in which $f \leq n$ processors crash. We present an *early-stopping* algorithm for lattice agreement whose performance is superior to early-stopping algorithms for consensus. More specifically, each nonfaulty processor decides within $\min\{1 + \mathit{joinheight}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\}), \lfloor (3 + \sqrt{8f+1})/2 \rfloor\}$ rounds, for any execution of the algorithm in which $f \leq n$ processors crash. In particular, this algorithm distinguishes itself from a comparable algorithm of Attiya *et al.* [5] that requires $\Omega(\lg n)$ communication rounds in *every* execution.

Keywords: Lattice agreement, fault-tolerance, distributed algorithms.

1 Introduction

The *lattice agreement* decision problem was introduced by Attiya, Herlihy and Rachman [5] in an effort to identify connections between implementing *concurrent objects* and solving *decision problems* in *wait-free* computation. Roughly speaking, in this problem, n processors start with input values drawn from a *lattice* $\mathcal{L}$, a special case of a partially ordered set, and must (non-trivially)

[1] Supported by funds for the promotion of research at University of Cyprus.

decide on output values that are comparable to each other in the lattice. Thus, the lattice agreement decision problem is a weakening of traditional *consensus* (see, e.g., [14, Chapter 12]), which, unlike consensus, *can* be solved in failure-prone asynchronous systems. The lattice agreement decision problem models situations arising in applications such as updating a distributed database, or detecting termination, deadlock, or a stable property of a distributed system. In such situations, processors need to adopt recent and consistent "views" of an execution. Such "views" capture a global snapshot of a distributed system, and processors may use them to infer possible future behaviors of the system.

Besides the fact that lattice agreement is an interesting decision problem in its own right, Attiya *et al.* show [5, Theorem 3.9] that, in the shared memory model of computation, solving the lattice agreement problem is equivalent to implementing the *atomic snapshot* object [1,3]; that is, given any solution to lattice agreement, it is possible to construct an implementation of a snapshot object, and vice versa. A snapshot object is a valuable tool that simplifies the design and verification of concurrent algorithms by restricting the possible interleavings of an execution (see, e.g., [6,9]); thus, an additional motivation to solve the lattice agreement problem stems from this equivalence: in order to implement a snapshot object in a given model of distributed computation for which the equivalence holds, it may be helpful to solve the lattice agreement problem in the specific model and reduce the solution to an implementation of the snapshot object.

Attiya *et al.* [5, Section 4] present an algorithm that solves lattice agreement in the synchronous, message-passing model of distributed computation, subject to crash failures; this algorithm is recursive, using a "branch-and-bound" technique, and terminates after $\lg n + 1$ communication rounds. In this work, we still consider the same model, and we assume that the lattice has a *unique* least element; this assumption is reasonable for using lattice agreement to implement an atomic snapshot, since in such an implementation, a lattice element corresponds to a vector of "round numbers," all of which are initially zero, that can grow without bound.

We present a new algorithm for lattice agreement, which distinguishes itself from the comparable algorithm of Attiya *et al.* in being *early-stopping* [8]; that is, its running time is bounded by the number of failures that actually occur in an execution, whereas the algorithm of Attiya *et al.* [5] requires $\Omega(\lg n)$ rounds in *every* execution. In particular, consider any execution in which $f \leq n$ processors crash, and assume that processors start with input values $X_1, X_2, \ldots, X_n$ (not necessarily distinct). Assume that one starts with the set of input values $\{X_1, X_2, \ldots, X_n\}$, and repeatedly enlarges this set by "inserting" other elements of the lattice that can be formed as *joins* of elements currently in the set; roughly speaking, the resulting set is a *chain* if any two of its elements can be "compared" in the lattice. Denote *joinheight*$(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$ the size of a *maximal* chain that can be produced in this way. We show that each non-faulty processor decides within $\min\{1 + \mathit{joinheight}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\}), \lfloor (3 + \sqrt{8f+1})/2 \rfloor\}$ rounds.

The rest of this paper is organized as follows. We provide our definitions in Section 2. The algorithm that solves the lattice agreement problem is presented and analyzed in Section 3. We conclude, in Section 4, with a discussion of our results and a look ahead to some possible future work.

2 Definitions

In Section 2.1, we define our model of computation. Lattices are introduced in Section 2.2, while Section 2.3 poses the lattice agreement problem. Both Sections 2.2 and 2.3 borrow from Attiya *et al.* [5, Section 2]. Throughout, denote for any integer $n \geq 2$, $[n] = \{1, 2, \ldots, n\}$.

2.1 Model of Computation

Our model of distributed computation is a standard synchronous, message-passing model subject to crash failures; we sketch the model here, and we refer the reader to [14, Chapter 6], or to previous work using this model [7,10,12,16], for more details. We consider a message-passing system with n processors denoted $p_1, p_2, \ldots, p_n$. We will sometimes use processor indices to denote processors. Each processor is modeled as a (possibly infinite) state machine.

Processors execute in lock-step, and an *execution* proceeds in a sequence of consecutively numbered *rounds*; the initial round is round 1. In each round, a processor may perform some local computation and send messages to any group of processors; the processors in that group are guaranteed to receive these messages before the next round. We assume that the state of processor p_i contains a special component buff_i in which incoming messages are buffered at each round, and removed by the next round.

We consider a mild form of failure where a processor may halt in the middle of an execution. If a processor crashes in a certain round, then only some (possibly empty) subset of the messages it sent during that round arrives. Furthermore, this processor will not participate in any of the subsequent rounds. A crashed processor is called *faulty*; processors that do not crash are called *nonfaulty*.

A *partially ordered set* is a (possibly infinite) set $\mathcal{L}$ with a *partial order* $\leq$. For any two elements $S_1, S_2 \in \mathcal{L}$, say that S_1 and S_2 are *comparable within* $\mathcal{L}$ *under* $\leq$, or *comparable* for short, if either $S_1 \leq S_2$ or $S_2 \leq S_1$; S_1 and S_2 are *incomparable* if they are not comparable. Write $S_1 < S_2$ if $S_1 \leq S_2$ but $S_1 \neq S_2$. A *chain* of $\mathcal{L}$ is a totally ordered subset of $\mathcal{L}$. The *height* of $\mathcal{L}$, denoted $\mathit{height}(\mathcal{L})$, is the size of a maximal chain of $\mathcal{L}$, or infinite if $\mathcal{L}$ has infinite chains.

For any (possibly empty) subset $\mathcal{S}$ of $\mathcal{L}$, say that $S \in \mathcal{L}$ is an *upper bound* of $\mathcal{S}$ if for each $S_i \in \mathcal{S}$, $S_i \leq S$. A *least upper bound,* or *join,* of $\mathcal{S}$, denoted $\mathit{join}(\mathcal{S})$, is an upper bound S of $\mathcal{S}$ such that if $\widehat{S}$ is an upper bound of $\mathcal{S}$, then $S \leq \widehat{S}$. A *lower bound* of $\mathcal{S}$ and a *greatest lower bound,* or *meet,* of $\mathcal{S}$, denoted $\mathit{meet}(\mathcal{S})$, are defined similarly. A *lattice* is a partially ordered set $\mathcal{L}$ such that for every (possibly empty) subset $\mathcal{S}$ of $\mathcal{L}$, join and meet of $\mathcal{S}$ exist. A *least element* of $\mathcal{L}$ is a meet of $\mathcal{L}$. We will assume that the lattice $\mathcal{L}$ has a *unique* least element, denoted $\mathbf{0}_{\mathcal{L}}$. (Lattices with no infinite chains have this property; see, e.g., [13, Chapter 23].)

For any (possibly empty) subset $\mathcal{S}$ of $\mathcal{L}$, we inductively define the *sublattice of* $\mathcal{L}$ *generated by* $\mathcal{S}$, denoted $\mathcal{L} \downarrow \mathcal{S}$, as follows:

(i) for each $S \in \mathcal{S}$, $S \in \mathcal{L} \downarrow \mathcal{S}$;
(ii) for any integer $l \geq 2$, if $S_{i_1}, S_{i_2}, \ldots, S_{i_l} \in \mathcal{L} \downarrow \mathcal{S}$, then
  (a) $\mathit{join}(\{S_{i_1}, S_{i_2}, \ldots, S_{i_l}\}) \in \mathcal{L} \downarrow \mathcal{S}$, and
  (b) $\mathit{meet}(\{S_{i_1}, S_{i_2}, \ldots, S_{i_l}\}) \in \mathcal{L} \downarrow \mathcal{S}$;
(iii) nothing is in $\mathcal{L} \downarrow \mathcal{S}$ unless it can be obtained by using rules (i) and (ii).

So, $\mathcal{L} \downarrow \mathcal{S}$ is the smallest sublattice of $\mathcal{L}$ including $\mathcal{S}$ (cf. [2, Exercise II.1.6]).

Roughly speaking, for any (possibly empty) subset $\mathcal{S}$ of $\mathcal{L}$, the joins of $\mathcal{L} \downarrow \mathcal{S}$ is the subset of $\mathcal{L} \downarrow \mathcal{S}$ that contains all elements that can "enter" $\mathcal{L} \downarrow \mathcal{S}$ as elements or $\mathcal{S}$ or as joins of other elements; formally, define the *joins of* $\mathcal{L} \downarrow \mathcal{S}$, denoted $\mathit{joins}(\mathcal{L} \downarrow \mathcal{S})$, as follows:

(i) for each $S \in \mathcal{S}$, $S \in \mathit{joins}(\mathcal{L} \downarrow \mathcal{S})$;
(ii) for any integer $l \geq 2$, if $S_{i_1}, S_{i_2}, \ldots, S_{i_l} \in \mathit{joins}(\mathcal{L} \downarrow \mathcal{S})$, then

$$\mathit{join}(\{S_{i_1}, S_{i_2}, \ldots, S_{i_l}\}) \in \mathit{joins}(\mathcal{L} \downarrow \mathcal{S})\,;$$

(iii) nothing is in $\mathit{joins}(\mathcal{L} \downarrow \mathcal{S})$ unless it can be obtained by using rules (i) and (ii).

We show that each element of $\mathit{joins}(\mathcal{L} \downarrow \mathcal{S})$ is the join of some subset of $\mathcal{S}$.

Proposition 1 *For each $S \in joins(\mathcal{L} \downarrow \mathcal{S})$, $S = join(\mathcal{T})$ for some set $\mathcal{T} \subseteq \mathcal{S}$.*

Moreover, if $S = join(\widehat{\mathcal{T}})$ for some set $\widehat{\mathcal{T}}$ such that for each $\tau_i \in \widehat{\mathcal{T}}$, $\tau_i = join(\mathcal{T}_i)$ for some set $\mathcal{T}_i \subseteq \mathcal{S}$, then $S = join(\cup_i \mathcal{T}_i)$.

Proof. By induction on the number of applications of rule (ii) required for S to enter $joins(\mathcal{L} \downarrow \mathcal{S})$.

For the base case, where zero applications of rule (ii) are required, S enters $joins(\mathcal{L} \downarrow \mathcal{S})$ by rule (i). Then, $S = S_i$ for some $S_i \in \mathcal{S}$. Since $S_i = join(\{S_i\})$, the claim follows.

Assume now that a nonzero number of applications of rule (ii) is required for S to enter $joins(\mathcal{L} \downarrow \mathcal{S})$; thus, $S = join(\widehat{\mathcal{T}})$ where for each $\tau_i \in \widehat{\mathcal{T}}$, $\tau_i \in joins(\mathcal{L} \downarrow \mathcal{S})$. Assume inductively that for each $\tau_i \in \widehat{\mathcal{T}}$, $\tau_i = join(\mathcal{T}_i)$ for some set $\mathcal{T}_i \subseteq \mathcal{S}$.

Since the join is an upper bound, for each $\tau_i \in \widehat{\mathcal{T}}$, $\tau_i \leq S$, and τ_i is an upper bound of $\mathcal{T}_i$. Hence, by transitivity, S is an upper bound of $\mathcal{T}_i$, which implies that S is an upper bound of $\cup_i \mathcal{T}_i$. By definition of join, it follows that $join(\cup_i \mathcal{T}_i) \leq S$.

By definition of join, $join(\cup_i \mathcal{T}_i)$ is an upper bound of $\cup_i \mathcal{T}_i$; since $\mathcal{T}_i \subseteq \cup_i \mathcal{T}_i$, it follows that $join(\cup_i \mathcal{T}_i)$ is an upper bound of $\mathcal{T}_i$, so that, by definition of join, $join(\mathcal{T}_i) \leq join(\cup_i \mathcal{T}_i)$. Thus, $join(\cup_i \mathcal{T}_i)$ is an upper bound of $\cup_i \{join(\mathcal{T}_i)\} = \cup_i \{\tau_i\} = \mathcal{T}$. Since S is the least upper bound of $\mathcal{T}$, this implies that $S \leq join(\cup_i \mathcal{T}_i)$. Hence, $S = join(\cup_i \mathcal{T}_i)$, as needed. □

The *joinheight* of $\mathcal{L} \downarrow \mathcal{S}$, denoted $joinheight(\mathcal{L} \downarrow \mathcal{S})$, is the height of $joins(\mathcal{L} \downarrow \mathcal{S})$.

2.3 The Lattice Agreement Problem

In the *lattice agreement* problem [5], each processor p_i is assigned some input X_i, and must decide on some output Y_i. Both input and output values are drawn from a lattice $\mathcal{L}$ with partial order $\leq$. An algorithm *solves lattice agreement* if it satisfies the following three conditions:

- *Comparability:* for all indices $i, j \in [n]$, Y_i and Y_j are comparable;
- *Downward-Validity:* for all indices $i \in [n]$, $X_i \leq Y_i$;
- *Upward-Validity:* for all indices $i \in [n]$, $Y_i \leq join(\{X_1, X_2, \ldots, X_n\})$.

The comparability condition requires that outputs of processors are all comparable to each other within the lattice. The downward-validity condition requires that the output of each processor is not smaller in the lattice than its input. The upward-validity condition requires that the output of each processor is not greater in the lattice than the join of all the inputs.

An algorithm that solves lattice agreement is *wait-free* (cf. [11]) if, for each of its executions, every nonfaulty processor decides within a bounded number of rounds, regardless of the execution or failures of other processors; say that it *solves lattice agreement in r rounds* if every nonfaulty processor decides no later than round r. An algorithm that solves lattice agreement is *early-stopping* (cf. [8]) if for each execution in which f processors crash, every nonfaulty processor decides after running for $O(f)$ rounds. Clearly, any early-stopping algorithm is also wait-free.

3 The Algorithm

In this section, we present our main result.

Theorem 2 *There is an early-stopping algorithm that solves lattice agreement in* $\min\{1+\mathit{joinheight}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\}), \lfloor (3+\sqrt{8f+1})/2 \rfloor\}$ *rounds, for any execution in which processors $p_1, p_2, \ldots, p_n$ start with input values $X_1, X_2, \ldots, X_n$, respectively, and f processors crash.*

In Section 3.1, we provide a description of an algorithm $\mathcal{A}$ with the claimed properties. A correctness proof and analysis of round complexity for $\mathcal{A}$ are presented in Sections 3.2 and 3.3, respectively.

3.1 Description and Preliminaries

The local state of processor p_i contains components S_i and r_i; the component S_i represents the "current decision value," while the component r_i holds a nonnegative integer *round number,* initially 1.

Roughly speaking, a processor changes its current decision value in a round only if some value received in the previous round is incomparable to its current decision value. In round 1, if $X_i = \mathbf{0}_{\mathcal{L}}$, then p_i decides on $\mathbf{0}_{\mathcal{L}}$ and halts, else p_i adopts X_i as its current decision value S_i and broadcasts it. In round $r > 1$, p_i checks if any of the values received in round $r-1$ is incomparable to S_i. If so, then S_i is replaced by its join with all values received in round $r-1$, and p_i broadcasts S_i and passes to round $r+1$; else, p_i decides on S_i and halts.

Precondition: initial next-phase transition

$r_i = 1$

$X_i \neq \mathbf{0}_{\mathcal{L}}$

Effect:

$S_i := X_i$

$\mathbf{broadcast}(S_i)$

$r_i := r_i + 1$

Precondition: initial decision transition

$r_i = 1$

$X_i = \mathbf{0}_{\mathcal{L}}$

Effect:

$\mathbf{decide}(\mathbf{0}_{\mathcal{L}})$

Precondition: next-phase transition

$r_i > 1$

for some $R_j \in \mathit{buff}_i$, $S_i \not\leq R_j$ and $R_j \not\leq S_i$

Effect:

$S_i := \mathit{join}(\{S_i\} \cup \{R_j \mid R_j \in \mathit{buff}_i\})$

$\mathbf{broadcast}(S_i)$

$r_i := r_i + 1$

Precondition: decision transition

$r_i > 1$

for every $R_j \in \mathit{buff}_i$, either $S_i \leq R_j$ or $R_j \leq S_i$

Effect:

$\mathbf{decide}(S_i)$

Fig. 1. The algorithm $\mathcal{A}$: program for processor p_i

Figure 1 presents the code for processor p_i in a precondition-effect style that is commonly used to describe I/O automata [15]. A $\mathbf{decide}(Y)$ operation causes p_i to enter a decision state for value Y (by recording the decision in the appropriate state component); a $\mathbf{broadcast}(S)$ operation causes p_i to send the message S to all other processors.

For each nonfaulty processor p_i, define the *decision round* of p_i, denoted ρ_i, to be the round in which p_i decides. For the case where $\rho_i > 1$, consider the sequence $S_i^{(2)}, \ldots, S_i^{(\rho_i)}$ of values held by S_i, where for each r, $2 \leq r \leq \rho_i$, $S_i^{(r)}$ is the value held by S_i right before p_i executes round r. The next result summarizes certain properties of the sequence $S_i^{(2)}, \ldots, S_i^{(\rho_i)}$; these properties will be crucial in both showing correctness for and analyzing the round complexity of $\mathcal{A}$.

Lemma 3 *For each nonfaulty processor p_i such that $\rho_i > 1$,*

(1) $S_i^{(2)} = X_i$ *and* $S_i^{(\rho_i)} = Y_i$*;*

(2) $S_i^{(2)} < \ldots < S_i^{(\rho_i)}$;
(3) *for each* r, $2 \leq r \leq \rho_i$, $S_i^{(r)} \in \mathit{joins}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$.

Proof. Property (1) follows immmediately from the algorithm (see initial next-phase transition and decision transition in Figure 1).

To show (2), consider any consecutive $S_i^{(r-1)}$ and $S_i^{(r)}$, where $2 < r \leq \rho_i$. By the algorithm, $S_i^{(r)}$ is the least upper bound of $S_i^{(r-1)}$ and all values received by p_i at the end of round $r-1$; thus, $S_i^{(r-1)} \leq S_i^{(r)}$. By the algorithm, there is some value R_j received by p_i at the end of round $r-1$ that is incomparable to $S_i^{(r-1)}$; since $R_j \leq S_i^{(r)}$, it follows that $S_i^{(r-1)} \neq S_i^{(r)}$. Hence, $S_i^{(r-1)} < S_i^{(r)}$, as needed.

We continue to show (3) by induction on r. For the base case where $r = 2$, $S_i^{(1)} = X_i$ by (1), and the claim holds trivially. Assume inductively that the claim holds for all rounds $2, \ldots, r-1$, and consider round r. By induction hypothesis, both $S_i^{(r-1)}$ and each of $S_j^{(r-1)}$ are in $\mathit{joins}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$. By the algorithm, $S_i^{(r)} = \mathit{join}(\{S_i^{(r-1)}\} \cup \{S_j^{(r-1)} \mid S_j^{(r-1)} \in \mathit{buff}_i\})$. It follows, by rule (ii)(a) used in defining the $\mathit{joins}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$ that $S_i^{(r)} \in \mathit{joins}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$, as needed. $\square$

3.2 Correctness

We show that processors' decisions satisfy the three conditions in the definition of the lattice agreement problem (Section 2.3).

We first show comparability. Consider nonfaulty processors p_i and p_j, and assume, without loss of generality, that p_i decides no later than p_j, i.e., $\rho_i \leq \rho_j$. If $\rho_i = 1$, then, by the algorithm, $Y_i = \mathbf{0}_{\mathcal{L}}$, so that Y_i and Y_j are trivially comparable, since $\mathbf{0}_{\mathcal{L}}$ is the least element of $\mathcal{L}$. So assume $\rho_i > 1$. By the algorithm, p_i broadcasts $S_i^{(\rho_i)}$ in round $\rho_i - 1$. There are two possibilities regarding the values received by p_j in round ρ_i:

(i) All of these values are comparable to $S_j^{(\rho_i)}$; in particular, $Y_i = S_i^{(\rho_i)}$ and $S_j^{(\rho_i)}$ are comparable. Then, by the algorithm, p_j decides on $Y_j = S_j^{(\rho_i)}$ in round ρ_i, and comparability holds.
(ii) Some of these values is incomparable to $S_j^{(\rho_i)}$, so that p_j does not decide in round ρ_i, i.e., $\rho_i < \rho_j$. By the algorithm, $S_j^{(\rho_i+1)}$ is the join of $S_j^{(\rho_i)}$ with all values received by p_j in round ρ_i; in particular, $S_i^{(\rho_i)} \leq S_j^{(\rho_i+1)}$. Since $\rho_i + 1 \leq \rho_j$, Lemma 3(2) implies that $S_j^{(\rho_i+1)} < S_j^{(\rho_j)} = Y_j$. It follows that

$Y_i = S_i^{(\rho_i)} < Y_j$, and comparability holds.

We continue to show downward-validity. Consider any nonfaulty processor p_i. We proceed by case analysis on the decision round of p_i. Assume first that $\rho_i = 1$, so that p_i decides on $\mathbf{0}_{\mathcal{L}}$; since, by the algorithm, p_i decides on $\mathbf{0}_{\mathcal{L}}$ only if its input equals $\mathbf{0}_{\mathcal{L}}$, downward-validity holds trivially. Assume now that $\rho_i > 1$. By Lemma 3(1) and (2), $X_i = S_i^{(2)} < \ldots < S_i^{(\rho_i)} = Y_i$, and downward-validity holds.

We finally show upward-validity. Consider any nonfaulty processor p_i. We proceed by case analysis on the decision round of p_i. Assume first that $\rho_i = 1$, so that p_i decides on $\mathbf{0}_{\mathcal{L}}$; then, upward-validity holds trivially since $\mathbf{0}_{\mathcal{L}}$ is the least element of $\mathcal{L}$. Assume now that $\rho_i > 1$. By Lemma 3(1), $Y_i = S_i^{(\rho_i)}$. It follows by Lemma 3(3) that $Y_i \in \mathit{joins}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$. Thus, by Proposition 1, $Y_i = \mathit{join}(\{X_{i_1}, \ldots, X_{i_l}\})$, where $\{X_{i_1}, \ldots, X_{i_l}\} \subseteq \{X_1, X_2, \ldots, X_n\}$. It follows that $Y_i \leq \mathit{join}(\{X_1, X_2, \ldots, X_n\})$, as needed.

3.3 Round Complexity

In this section, we prove an upper bound on the number of rounds incurred by the algorithm $\mathcal{A}$ in the worst case; this will establish the wait-freedom (and, thereby, the termination) of this algorithm.

Consider processor p_i deciding on Y_i in round $\rho_i > 1$. By Lemma 3(1), $Y_i = S_i^{(\rho_i)}$. By Lemma 3(3), for each r, $1 \leq r \leq \rho_i$, $S_i^{(\rho_i)} \in \mathit{joins}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$. Thus, it follows by Lemma 3(2) that the sequence of length $\rho_i - 1$ $S_i^{(2)}, \ldots, S_i^{(\rho_i)}$ forms a chain of $\mathit{joins}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$. Since the size of a maximal chain of $\mathit{joins}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$ is $\mathit{joinheight}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$, this implies that $\rho_i - 1 \leq \mathit{joinheight}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$, so that:

Lemma 4 *$\mathcal{A}$ solves lattice agreement in $1 + \mathit{joinheight}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$ rounds, for any execution in which processors $p_1, p_2, \ldots, p_n$ start with input values $X_1, X_2, \ldots, X_n$, respectively.*

We continue to show an upper bound on the number of rounds taken by $\mathcal{A}$, which is a function of the number of failures f occurring in an execution.

Lemma 5 *$\mathcal{A}$ solves lattice agreement in $\lfloor (3 + \sqrt{8f+1})/2 \rfloor$ rounds, for any execution in which f processors crash.*

Proof. Consider any execution α of $\mathcal{A}$ in which f processors crash; denote $f_r \leq f$ the number of processors that crash in round $r \geq 1$. We show:

Claim 6 *In α, for any round $r_0 \geq 1$, every processor decides within $r_0+f_{r_0}+1$ rounds.*

Proof. Without loss of generality, let $1, \ldots, f_{r_0}$ be the processors crashing in round r_0 of α. Clearly, by Lemma 3, for any nonfaulty processor p_i, for each r, $r_0+1 \leq r \leq \rho_i$, $S_i^{(r)} \in joins(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\})$. Thus, by Proposition 1 and the structure of the algorithm, for each r, $r_0+1 \leq r \leq \rho_i$, $S_i^{(r)} = join(\{X_{f_{r_0}+1}, \ldots, X_n\} \cup \{X_{i_1}, \ldots, X_{i_k}\})$, where $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, f_{r_0}\}$. Since, by the algorithm, p_i does not decide in round r only if it updates $S_i^{(r-1)}$, the maximum number of rounds p_i can remain undecided after it completes round r_0, is at most the length of the longest possible sequence $S_i^{(r_0+1)}, \ldots, S_i^{(\rho_i)}$. Since, by Lemma 3, $S_i^{(r_0+1)} < \ldots < S_i^{(\rho_i)}$, this longest possible sequence is the following sequence of length $f_{r_0}+1$:

- $join(\{X_{f_{r_0}+1}, X_2, \ldots, X_n\}$,
- $join(\{X_{f_{r_0}+1}, X_2, \ldots, X_n\} \cup \{X_{i_{\pi(1)}}\})$,
- $join(\{X_{f_{r_0}+1}, \ldots, X_n\} \cup \{X_{i_{\pi(1)}}, X_{i_{\pi(2)}}\})$,
- $\ldots$,
- $join((\{X_{f_{r_0}+1}, \ldots, X_n\} \cup \{X_{i_{\pi(1)}}, \ldots, X_{i_{\pi(f_{r_0})}}\})$,

where π is any permutation of $\{1, \ldots, f_{r_0}\}$. That is, the $f_{r_0}+1$ elements of the sequence are those obtained by joining in $0, 1, \ldots$ and f_{r_0} elements from $X_1, \ldots, X_{f_{r_0}}$. Thus, the total number of rounds for p_i to decide is no more than r_0 (for rounds up to round r_0) plus $f_{r_0}+1$, the number of rounds needed subsequently, which is $r_0+f_{r_0}+1$, as needed. $\Box$

Assume that $\mathcal{A}$ solves lattice agreement in ℓ rounds, for any execution in which f processors crash. Clearly, ℓ is no more than the upper bounds established in Claim 6 for any such execution. Thus, for each index r_0, $1 \leq r_0 < \ell$, $\ell \leq r_0+f_{r_0}+1$, so that

$$\sum_{r_0=1}^{\ell-1}(\ell-r_0) \leq \sum_{r_0=1}^{\ell-1}(f_{r_0}+1) \;=\; \sum_{r_0=1}^{\ell-1} f_{r_0} + \sum_{r_0=1}^{\ell-1} 1 \;\leq\; f+\ell-1\,,$$

or $\sum_{r_0=1}^{\ell-1} r_0 \leq f+\ell-1$, or $(\ell-1)\ell/2 \leq f+\ell-1$, implying that $\ell^2-3\ell-2f+2 \leq 0$. Thus, ℓ may not exceed the positive root of the quadratic form in the left side, so that $\ell \leq (3+\sqrt{9-4(-2f+2)})/2 = (3+\sqrt{8f+1})/2$. Since ℓ is an integer, this implies that $\ell \leq \lfloor(3+\sqrt{8f+1})/2\rfloor$, as needed. $\Box$

Lemmas 4 and 5 together imply:

Proposition 7 *Algorithm $\mathcal{A}$ solves lattice agreement in* $\min\{1+\mathit{joinheight}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\}), \lfloor (3+\sqrt{8f+1})/2 \rfloor\}$ *rounds, for any execution in which processors $p_1, p_2, \ldots, p_n$ start with input values $X_1, X_2, \ldots, X_n$, respectively, and f processors crash.*

4 Discussion

We have presented a synchronous, early-stopping algorithm for lattice agreement in the message-passing model of distributed computation. Each processor decides using no more than $\min\{1 + \mathit{joinheight}(\mathcal{L} \downarrow \{X_1, X_2, \ldots, X_n\}), \lfloor (3 + \sqrt{8f+1})/2 \rfloor\}$ rounds, for any execution in which n processors, out of which f crash, start with input values $X_1, X_2, \ldots, X_n$. The translation of this algorithm to the synchronous shared memory model subject to crash failures is straightforward.

The most obvious open question left open by our work is whether this upper bound is tight or not; does there exist an early-stopping algorithm that solves lattice agreement in the synchronous, message-passing model in $o(\sqrt{f})$ rounds? Also, can our synchronous algorithm be extended to yield a *wait-free* and early-stopping algorithm for lattice agreement in the completely asynchronous model (see, e.g., [14, Chapter 21])? (Attiya *et al.* [5, Section 5] show that their synchronous algorithm can be extended to yield a corresponding asynchronous, wait-free lattice agreement algorithm.) It would also be interesting to study the lattice agreement problem in the partially synchronous message-passing model of computation (see, e.g., [14, Chapter 25]), in the presence of crash (or even more severe) processor failures.

Some more recent results on lattice agreement in the shared, read/write memory model of computation appear in [4].

Acknowledgments:

We are indebted to one anonymous referee for numerous valuable comments, corrections and suggestions.

References

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt and N. Shavit, "Atomic Snapshots of Shared Memory," *Journal of the ACM,* Vol. 40, No. 4, pp. 873–890, September 1993.

[2] M. Aigner, *Combinatorial Theory,* Springer-Verlag, 1979.

[3] J. Anderson, "Composite Registers," *Distributed Computing,* Vol. 6, No. 3, pp. 141–154, 1993.

[4] H. Attiya and A. Fouren, "Adaptive Wait-free Algorithms for Lattice Agreement and Renaming," *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing,* pp. 277–286, June/July 1998.

[5] H. Attiya, M. Herlihy and O. Rachman, "Atomic Snapshots Using Lattice Agreement," *Distributed Computing,* Vol. 8, pp. 121–132, 1995.

[6] H. Attiya, N. Lynch and N. Shavit, "Are Wait-Free Algorithms Fast?" *Journal of the ACM,* Vol. 41, No. 4, pp. 725–763, July 1994.

[7] S. Chaudhuri, "Towards a Complexity Hierarchy of Wait-Free Concurrent Objects," *Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing,* pp. 730–737, October 1991.

[8] D. Dolev, R. Reischuk, and H. R. Strong, "Early Stopping in Byzantine Agreement," *Journal of the ACM,* Vol. 37, No. 4, pp. 720–741, October 1990.

[9] R. Gawlick, N. Lynch and N. Shavit, "Concurrent Time-Stamping Made Simple," *Proceedings of the 1st Israel Symposium on Theory of Computing and Systems,* Lecture Notes in Computer Science, Vol. 601, pp. 171–185, Springer-Verlag, May 1992.

[10] J. Halpern and Y. Moses, "Knowledge and Common Knowledge in a Distributed Environment," *Journal of the ACM,* Vol. 37, No. 3, pp. 549–587, July 1990.

[11] M. Herlihy, "Wait-free Synchronization," *ACM Transactions on Programming Languages and Systems,* Vol. 13, No. 1, pp. 124–149, January 1991.

[12] M. Herlihy and M. Tuttle, "Wait-Free Computation in Message-Passing Systems," *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing,* pp. 347–362, August 1990.

[13] J. H. van Lint and R. M. Wilson, *A Course in Combinatorics,* Cambridge University Press, 1992.

[14] N. Lynch, *Distributed Algorithms,* Morgan Kaufmann, 1996.

[15] N. Lynch and M. Tuttle, "An Introduction to Input/Output Automata," *CWI Quarterly,* Vol. 2, No. 3, pp. 219–246, September 1989.

[16] Y. Moses and M. Tuttle, "Programming Simultaneous Actions Using Common Knowledge," *Algorithmica,* Vol. 3, No. 1, pp. 121–169, 1988.