The Step Complexity of Multidimensional Approximate Agreement

Hagit Attiya
Department of Computer Science, Technion, Israel

Faith Ellen
Department of Computer Science, University of Toronto, Canada

Abstract

Approximate agreement allows a set of n processes to obtain outputs that are within a specified distance $\epsilon > 0$ of one another and within the convex hull of the inputs.

When the inputs are real numbers, there is a wait-free shared-memory approximate agreement algorithm [16] whose step complexity is in $O(n \log(S/\epsilon))$, where S, the *spread* of the inputs, is the maximal distance between inputs. There is another wait-free algorithm [17] that avoids the dependence on n and achieves $O(\log(M/\epsilon))$ step complexity where M, the *magnitude* of the inputs, is the absolute value of the maximal input.

This paper considers whether it is possible to obtain an approximate agreement algorithm whose step complexity depends on neither n nor the magnitude of the inputs, which can be much larger than their spread. On the negative side, we prove that $\Omega\left(\min\left\{\frac{\log M}{\log\log M}, \frac{\sqrt{\log n}}{\log\log n}\right\}\right)$ is a lower bound on the step complexity of approximate agreement, even when the inputs are real numbers. On the positive side, we prove that a polylogarithmic dependence on n and S/ϵ can be achieved, by presenting an approximate agreement algorithm with $O(\log n(\log n + \log(S/\epsilon)))$ step complexity. Our algorithm works for multidimensional domains. The step complexity can be further restricted to be in $O(\min\{\log n(\log n + \log(S/\epsilon)), \log(M/\epsilon)\})$ when the inputs are real numbers.

2012 ACM Subject Classification Theory of computation → Shared memory algorithms; Theory of computation → Distributed algorithms

Keywords and phrases approximate agreement, conflict detection, shared memory, wait-freedom, step complexity

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2022.6

Funding *Hagit Attiya*: Supported by the Israel Science Foundation (grants 380/18 and 1425/22).
Faith Ellen: Supported by the Natural Science and Engineering Research Council of Canada (grant RGPIN-2020-04178).

Acknowledgements We thank Sasho Nikolov for useful discussion. We also appreciate the helpful comments of the anonymous reviewers.

1 Introduction

Approximate agreement allows a set of n processes, each starting with an input from a domain, to obtain outputs (in the same domain) that are close to each other and in the convex hull of the inputs. A parameter ϵ represents an upper bound on how close the outputs are. Originally, Dolev, Lynch, Pinter, Stark and Weihl [7] considered the one-dimensional case, where the domain is $\mathbb{R}$, the *real numbers*, and motivated the problem by clock synchronization and the stabilization of inputs from sensors. More recently, Mendes, Herlihy, Vaidya, and Garg [14, 15, 18] considered *multidimensional* approximate agreement, also called *approximate vector consensus*, where the domain of inputs and outputs is $\mathbb{R}^k$, for some integer $k \geq 2$. The multidimensional variant was motivated by distributed algorithms for optimization problems.

© Hagit Attiya and Faith Ellen;
licensed under Creative Commons License CC-BY 4.0

Wait-free shared-memory algorithms to reach approximate agreement for asynchronous processes that may fail by crashing are evaluated by their (individual) *step complexity*, that is, the maximum number of reads and writes to shared memory a process performs until it obtains an output. There is a wait-free one-dimensional approximate agreement algorithm with $O(\log(S/\epsilon))$ iterations due to Moran [16], where S, the *spread* of the inputs, is the maximum distance between any two inputs. Each iteration requires one *update* and one *scan* of an atomic snapshot object [2]. Using Attiya and Rachman's implementation of a snapshot object from *single-writer* registers [6], this leads to $O(n \log n \log(S/\epsilon))$ step complexity and, using Inoue, Masuzawa, Chen, and Tokura's implementation from *multi-writer* registers [13], this leads to $O(n \log(S/\epsilon))$ step complexity.

A simple argument shows that $\Omega(n)$ is a lower bound on the step complexity of approximate agreement using single-writer registers [4]. The dependence on n can be avoided by using multi-writer registers: An algorithm by Schenk [17] for inputs in $\mathbb{R}$ achieves $O(\log(M/\epsilon))$ step complexity, where M, the *magnitude* of the inputs, is the largest of the absolute values of the inputs.

The spread, S, of a set of inputs is at most twice its magnitude, M, but the magnitude might be significantly larger than the spread. This raises the challenge, addressed in this paper, of obtaining a wait-free approximate agreement algorithm whose step complexity depends on S, but not on n or M. If the same upper and lower bounds on the domain of the input values are known to all processes, then the bounded version of Schenk's algorithm has $O(\log(R/\epsilon))$ step complexity, where R is the difference between these two bounds. However, R can also be much larger than the spread of the inputs.

This paper provides both negative and positive answers. On the negative side, we prove that $\Omega\left(\min\left\{\frac{\log M}{\log\log M}, \frac{\sqrt{\log n}}{\log\log n}\right\}\right)$ is a lower bound on the step complexity of approximate agreement, even using multi-writer registers. Thus, for values of M that are larger than n (actually, even for $M \in 2^{\Omega(\sqrt{\log n})}$), step complexity that depends on n cannot be avoided. This lower bound is proved by reduction from the *conflict detection* problem [3]. We also prove a lower bound of $\frac{1}{2}\log_{\sqrt{2}+1}(S/\epsilon)$ on the step complexity of multidimensional approximate agreement using multi-writer registers. This extends and improves Herlihy's lower bound for one-dimensional approximate agreement among two or more processes using single-writer registers [11]. Related lower bounds were proved by Attiya, Lynch and Shavit [5] and Hoest and Shavit [12]. Ellen, Gelashvili, and Zhu [8] gave a lower bound on the number of registers needed to solve approximate agreement from a lower bound on its wait-free step complexity.

On the positive side, we prove that a polylogarithmic dependence on n can be achieved, by presenting a multidimensional approximate agreement algorithm with $O(\log n(\log n + \log(S/\epsilon)))$ step complexity. The algorithm repeatedly solves *two group approximate agreement*: combining one group of processes whose values are close together with another such group to form a larger group whose values can be slightly further apart. Two group approximate agreement is solved with a variant of approximate agreement, where processes may know slightly different domains for their inputs. A key step is showing how to solve this subproblem using Schenk's approximate agreement algorithm for inputs in [0,1].

In the one-dimensional case, we can run Schenk's algorithm in parallel with our new algorithm to achieve the best of both algorithms: an approximate agreement algorithm whose step complexity is $O(\min\{\log n(\log n + \log(S/\epsilon)), \log(M/\epsilon)\})$.

The one-dimensional problem was initially studied in message-passing systems [7]. Many different approximate agreement algorithms were subsequently developed for these models. Most of these algorithms are asynchronous and tolerate Byzantine failures. A good example is an approximate agreement algorithm that tolerates f Byzantine failures, when $n > 3f$ [1].

This algorithm works in $O(\log(S/\epsilon))$ (asynchronous) rounds. In each round, each process obtains information from $n-f$ processes. In the shared-memory model, this would translate into an algorithm with $O(n\log(S/\epsilon))$ step complexity.

Multidimensional approximate agreement was previously studied only in message-passing systems. The first algorithm for this problem in $\mathbb{R}^d$ uses $O(d\log(dS/\epsilon))$ rounds [15]. Later, dependency on d was eliminated: there is an algorithm by Függer and Nowak [9] that uses $O(\log(S/\epsilon))$ rounds. As with one-dimensional approximate agreement, this would translate into $O(n\log(S/\epsilon))$ step complexity. Results of Függer, Nowak and Schwarz [10] imply a lower bound of $\Omega(\log(S/\epsilon))$ on the number of rounds for solving multidimensional approximate agreement among two processes. Note that, like all prior lower bounds, their lower bound does not depend on the magnitude of the inputs, but only on their spread.

2 Model

We consider a system where n deterministic asynchronous processes, $p_0,\ldots,p_{n-1}$, communicate by reading and writing to shared multi-reader, multi-writer registers. In this model, all processes can read from and write to all registers. A *configuration* consists of the state of every process and the value of every shared register. In an *initial configuration*, each process starts in an initial state, which includes its input value, and all registers contain an initial value. Each process can be modelled as a deterministic state machine, specifying an algorithm that the process follows until it outputs a value.

Two configurations C and C' are *indistinguishable to process p_i* if p_i has the same state and all shared registers have the same values in both configurations.

A process p_i is *active in configuration C* if process p_i has not yet output a value. In this case, configuration Cp_i is the configuration reached from C when p_i performs the next step of its algorithm.

A *schedule* is a (finite or infinite) sequence of processes, specifying the order in which processes take steps. A non-empty schedule $\sigma = p_{i_1}p_{i_2}\cdots$ is *applicable* to a configuration C if process p_{i_1} is active in C and, for every prefix $p_{i_1}\cdots p_{i_k}$ of σ of length $k\geq 2$, process p_{i_k} is active in configuration $C_{k-1} = Cp_{i_1}\cdots p_{i_{k-1}}$. Suppose σ is a schedule that is applicable to configuration C. If σ has length k, then $C, p_{i_1}, C_1, \ldots, p_{i_k}, C_k$ is the *execution from C induced by σ*, where $C_k = Cp_{i_1}\cdots p_{i_k}$. If σ is an infinite schedule, then $C, p_{i_1}, C_1, \ldots$ is the *execution from C induced by σ*. The *solo execution of process p_i from C* is the execution induced by the longest schedule containing only process p_i that is applicable to C.

We assume each process p_i starts with an input value $x_i \in \mathbb{R}^d$ and, after performing some number of steps according to its algorithm, produces an output value $y_i \in \mathbb{R}^d$. An algorithm (or, more precisely, an algorithm for each process) *solves multidimensional approximate agreement with parameter ϵ* if (a) the distance between output values is at most ϵ, and (b) the output values are in the convex hull of the input values. An algorithm is *wait-free* if it ensures that every process that does not crash terminates within a finite number of its own steps.

The (individual) *step complexity* of an algorithm is the maximum number of steps taken by any one process in any possible execution of the algorithm.

3 A Multidimensional Approximate Agreement Algorithm

In our multidimensional approximate agreement algorithm, described in Section 3.4, processes repeatedly solve instances with increasing values of the accuracy parameter among increasingly many processes, using their output from one instance as their input to the next instance. The algorithm begins by solving instances among pairs of of processes. This produces groups of (two) processes whose inputs are close to one another. These groups are repeatedly combined, two at a time, in a tree-like manner, creating larger groups whose inputs can be slightly further apart.

The subproblem of combining two groups (where, in each group, the inputs are close together) is a restricted version of multidimensional approximate agreement, which we call *two group approximate agreement*. In Section 3.3, we show how to solve two group approximate agreement using another variant of multidimensional approximate agreement, which we call *approximate agreement with domain uncertainty*. This problem is defined in Section 3.2. There, we reduce it to a variant of approximate agreement for inputs in $[0, 1]$, which was introduced and efficiently solved by Schenk [17] and is described in Section 3.1.

3.1 Schenk's Algorithm

Schenk's wait-free approximate agreement algorithm r-agree(x, ϵ) [17] assumes that each process p_i has an input $x_i \in [0, 1]$ and all processes have a common accuracy parameter $\epsilon > 0$. It ensures that each non-faulty process outputs a value y_i such that $\min\{x_1, \ldots, x_n\} \leq y_i \leq \max\{x_1, \ldots, x_n\}$ and all outputs are within distance ϵ of one another.

When $\epsilon = 1/2$, r-agree uses two single-bit multi-writer registers, which are both initially 0. Processes with inputs in the interval [0,1/2] write 1 to one of these registers and processes with inputs in the interval [1/2,1] write 1 to the other register. Processes with input 1/2 output 1/2. A process with any other input reads the register it didn't write to and, if it sees 1, it also outputs 1/2. Otherwise it outputs its input. All outputs will lie in the interval of size 1/2 corresponding to the register that is written to first.

When $\epsilon = 1/4$, the same approach can be applied to this interval, using the outputs as inputs, to obtain new outputs in a subinterval of size 1/4. The only difficulty is that processes with output 1/2, which is on the boundary between both intervals [0,1/2] and [1/2,1], don't know which interval this is. The solution is for these processes to participate in the subproblems for both these intervals. In at least one of these two subproblems, it will output 1/2, so it can use its output from the other subproblem. When $\epsilon = 1/2^k$, where $k > 1$, this is done k times, each time reducing the size of the interval containing the inputs by a factor of 2. More generally, this is done $\lceil \log_2(1/\epsilon) \rceil$ times.

A useful generalization of approximate agreement is to allow each process p_i to have a different value $\epsilon_i > 0$ for its accuracy parameter. For this problem, which is called *ϵ-unknown approximate agreement*, all non-faulty processes must output a value within distance $\max\{\epsilon_1, \ldots, \epsilon_n\}$ of each other. As in approximate agreement, each output must lie between the smallest and largest inputs. This problem can also be solved using r-agree with $O(\log(\max\{1/\epsilon_1, \ldots, 1/\epsilon_n\}))$ step complexity.

Schenk used ϵ-unknown approximate agreement to solve approximate agreement for any real valued inputs $x_1, \ldots, x_n$ with $O(\log(\max\{|x_1|, \ldots, |x_n|\}/\epsilon))$ step complexity: First, each process p_i finds a value r_i such that (1) the interval $[-r_i, r_i]$ contains at least one of the original inputs and (2) these values differ from one another by at most a factor of 2. Then the processes solve this bounded version of approximate agreement by mapping their inputs to $[0, 1]$ and solving approximate agreement within this interval using accuracy parameters that can differ by at most a factor of 2.

3.2 Approximate Agreement with Domain Uncertainty

We consider a closely related problem, *approximate agreement with domain uncertainty*. In this problem, each process p_i has two points $u_i, v_i \in \mathbb{R}^k$ and a point $x_i \in \mathbb{R}^k$ on the line segment between them, expressed as $x_i = u_i + t_i(v_i - u_i)$, where $t_i \in [0, 1]$. We call this line segment the *domain* for process p_i. The domains of all processes are assumed to be close to one another. Specifically, there exists a constant $\delta > 0$ known to all processes such that $||u_i - u_j|| \leq \delta$ and $||v_i - v_j|| \leq \delta$ for all processes p_i and p_j. Each process p_i that does not crash must produce an output $y_i = u_i + t'_i(v_i - u_i)$ on the line between u_i and v_i such that $\min\{t_1, \ldots, t_n\} \leq t'_i \leq \max\{t_1, \ldots, t_n\}$ and the difference between any two outputs is at most ϵ, which is known to all processes.

Algorithm 1 solves approximate agreement with domain uncertainty for $\epsilon \geq 5\delta$. If the size of the domain of process p_i is small, then it simply sets $y_i = x_i$. Otherwise, it uses r-agree to solve approximate agreement with input t_i and uses the result t'_i to determine its output $y_i = u_i + t'_i(v_i - u_i)$.

Algorithm 1 Code for a process with inputs $u, v \in \mathbb{R}^k$, $t \in [0, 1]$, and parameters ϵ and δ.

```
   ApproxAgreeDU($u, v, t, \epsilon, \delta$)
1: $s \leftarrow ||v - u||$
2: if $s \leq 2\delta$ then return $u + t(v - u)$
3: $\epsilon' \leftarrow \epsilon/5s$
4: $t' \leftarrow$ r-agree($t, \epsilon'$)
5: return $u + t'(v - u)$
```

Consider an execution where process p_i calls ApproxAgreeDU$(u_i, v_i, t_i, \epsilon, \delta)$ with $t_i \in [0, 1]$ for $1 \leq i \leq n$. If p_i outputs $y_i = u_i + t_i(v_i - u_i) \in \mathbb{R}^k$ on line 2, then t_i lies between $\min\{t_1, \ldots, t_n\}$ and $\max\{t_1, \ldots, t_n\}$ and y_i is a point on the line segment between u_i and v_i. If p_i outputs the point $y_i = u_i + t'_i(v_i - u_i) \in \mathbb{R}^k$ on line 5, then t'_i is the value output by r-agree on line 4. The specifications of ϵ-unknown approximate agreement ensure that $0 \leq \min\{t_1, \ldots, t_n\} \leq t'_i \leq \max\{t_1, \ldots, t_n\} \leq 1$, so y_i is a point on the line segment between u_i and v_i.

To prove that ApproxAgreeDU is correct, it remains to show that all outputs are within ϵ of one another.

▶ **Lemma 1.** *For $1 \leq i, j \leq n$, if process p_i outputs $y_i = u_i + t'_i(v_i - u_i)$ and process p_j outputs $y_j = u_j + t'_j(v_j - u_j)$, then $||y_i - y_j|| \leq \epsilon$.*

Proof. For $1 \leq i \leq n$, let $s_i = ||u_i - v_i||$ be the size of the domain of process p_i. Since $||u_i - u_j|| \leq \delta$ and $||v_i - v_j|| \leq \delta$, it follows from the triangle inequality that $s_i = ||u_i - v_i|| = ||u_i - u_j + u_j - v_j + v_j - v_i|| \leq ||u_i - u_j|| + ||u_j - v_j|| + ||v_j - v_i|| \leq 2\delta + s_j$. By the triangle inequality,

$$\begin{aligned}
||y_i - y_j|| &= ||u_i + t'_i(v_i - u_i) - u_j - t'_j(v_j - u_j)|| \\
&= ||u_i - u_j + t'_i v_i - t'_i v_j + t'_i v_j - t'_j v_j + t'_j u_j - t'_i u_j + t'_i u_j - t'_i u_i|| \\
&\leq ||u_i - u_j|| + t'_i \cdot ||v_i - v_j|| + |t'_i - t'_j| \cdot ||v_j - u_j|| + t'_i \cdot ||u_j - u_i|| \\
&\leq \delta + 1 \cdot \delta + |t'_i - t'_j| \cdot ||v_j - u_j|| + 1 \cdot \delta \\
&= 3\delta + |t'_i - t'_j| \cdot s_j.
\end{aligned}$$

Let $I \subseteq \{1, \ldots, n\}$ be the set of identifiers of processes that perform line 4 and let $s' = \min\{s_m \mid m \in I\}$. Note that, by the test on line 2, if $I \neq \phi$, then $s' > 2\delta$. Hence $s_m \leq s' + 2\delta < 2s'$ for all $m \in I$. For each $m \in I$, let $\epsilon'_m = \epsilon/5s_m$, so $\max\{\epsilon'_m \mid m \in I\} = \max\{\epsilon/5s_m \mid m \in I\} = \epsilon/5 \min\{s_m \mid m \in I\} = \epsilon/5s'$.

First consider the case when $i, j \in I$. From the specifications of ϵ-unknown approximate agreement, $|t'_i - t'_j| \leq \max\{\epsilon'_m \mid m \in I\} = \epsilon/5s'$. Then $||y_i - y_j|| \leq 3\delta + |t'_i - t'_j| \cdot s_j \leq 3\epsilon/5 + (\epsilon/5s') \cdot 2s' = \epsilon$.

Otherwise, without loss of generality, suppose $j \notin I$. Then $s_j \leq 2\delta$ and $||y_i - y_j|| \leq 3\delta + |t'_i - t'_j| \cdot s_j \leq 3\delta + 1 \cdot 2\delta = 5\delta = \epsilon$. ◀

If $t_1 = \cdots = t_n = t$, then each nonfaulty process p_i outputs $u_i + t(v_i - u_i)$ since $t = \min\{t_1, \ldots, t_n\} \leq t'_i \leq \max\{t_1, \ldots, t_n\} = t$. In particular, if $t = 0$, then each nonfaulty process outputs its first argument and, if $t = 1$, then each nonfaulty process outputs its second argument.

3.3 Two Group Approximate Agreement

The *two group approximate agreement problem* is a restricted version of the approximate agreement problem in which the processes are divided into two groups, 0 and 1, such that, within each group, the inputs of the processes are guaranteed to be points in $\mathbb{R}^k$ that are within distance $\epsilon/5$ of one another. We will use ApproxAgreeDU (Algorithm 1) to solve this problem.

TwoGroupApproxAgree (Algorithm 2) uses two arrays of multi-writer registers $A[0..1]$ and $B[0..1]$, each with two components. The components of A are initially $\perp$ and can store any point in $\mathbb{R}^k$. The components of B are single bits and are initially 0. Only processes in group g write to component g of these arrays.

As in Schenk's approximate agreement algorithm, each process writes to one register ($A[g]$, where g is the group to which it belongs) and reads from the other register ($A[1-g]$). If a process in group g sees that $A[1-g]$ has not yet been written to, it informs the processes in group $1-g$ of this fact by writing 1 into $B[g]$ and reads from $A[1-g]$ again. If it sees that $A[1-g]$ has still not been written to, the process outputs its input.

Otherwise, the process participates in an instance of ApproxAgreeDU, using its input as one endpoint of its domain and the point it read as the other endpoint. The endpoints are ordered so that an input from group 0 is the first endpoint and an input from group 1 is the second point. Then the preconditions ensure that the domains of all processes are close to one another.

If a process in group g saw that $A[1-g]$ was first written to between its first and second reads, it uses its input for two group approximate agreement as its input point in this domain. However, if the process saw that $A[1-g]$ had been written to before its first read, it checks the bit $B[1-g]$. If it is 0, processes in the other group will participate in the instance of ApproxAgreeDU and the process also uses its input for two group approximate agreement as its input point in this domain. If it is 1, some processes in the other group may simply output their inputs. In this case, the process uses the other endpoint of its domain as its input point.

Consider an execution where TwoGroupApproxAgree(g_i, x_i, ϵ) is called by process p_i in group $g_i \in \{0, 1\}$, for $1 \leq i \leq n$. Furthermore, suppose $||x_i - x_j|| \leq \epsilon/5$ for every pair of processes p_i and p_j that are in the same group.

Algorithm 2 Code for a process in group G_g with input x.

```
    TwoGroupApproxAgree($g, x, \epsilon$)
1:  $a[g] \leftarrow x$
2:  $A[g] \leftarrow$ write($a[g]$)
3:  $a[1-g] \leftarrow$ read($A[1-g]$)
4:  if $a[1-g] = \perp$ then
5:      $B[g] \leftarrow$ write(1)
6:      $a[1-g] \leftarrow$ read($A[1-g]$)
7:      if $a[1-g] = \perp$ then
8:          return $x$
9:      else
10:         return ApproxAgreeDU($a[0], a[1], g, \epsilon, \epsilon/5$)
11: else
12:     $b \leftarrow$ read($B[1-g]$)
13:     if $b = 0$ then
14:         return ApproxAgreeDU($a[0], a[1], g, \epsilon, \epsilon/5$)
15:     else
16:         return ApproxAgreeDU($a[0], a[1], 1-g, \epsilon, \epsilon/5$)
```

▶ **Observation 2.** *The value 1 is written to most one component of B.*

Proof. Suppose the first step of the execution is by a process in group $1-g$. Then when any process from group g performs line 3, it does not see $\perp$ in $A[1-g]$ and, hence, it does not write 1 to $B[g]$ on line 5. ◀

Next, we show that the outputs are within the convex hull of the inputs.

▶ **Lemma 3.** *If process p_i outputs y_i, then y_i is in the convex hull of $\{x_1, \ldots, x_n\}$.*

Proof. If process p_i returns $y_i = x_i$ on line 8, the claim is true since x_i is in the convex hull of $\{x_1, \ldots, x_n\}$. Otherwise, y_i is the point returned by ApproxAgreeDU($a_i[0], a_i[1], t_i, \epsilon, \epsilon/5$), where $t_i \in \{0, 1\}$, g_i is the group to which p_i belongs, $a_i[g_i] = x_i$, and $a_i[1-g_i] \neq \perp$ is the value it read from $A[1-g_i]$ on line 3 or 6.

The only points written to $A[1-g_i]$ are elements of $\{x_1, \ldots, x_n\}$, so $a_i[1-g_i] \in \{x_1, \ldots, x_n\}$. From the specifications of ApproxAgreeDU, y_i is on the line segment between $a_i[0]$ and $a_i[1]$. Hence y_i is in the convex hull of $\{x_1, \ldots, x_n\}$. ◀

Finally, we show that all the outputs are sufficiently close to one another.

▶ **Lemma 4.** *Suppose that the inputs to all processes in group g are within $\epsilon/5$ of one another, for all $g \in \{0, 1\}$. For $1 \leq i, j \leq n$, if process p_i outputs the point y_i and process p_j outputs the point y_j, then $||y_i - y_j|| \leq \epsilon$.*

Proof. First consider the processes that return on lines 10, 14, or 16. Each such process p_i returns the result from ApproxAgreeDU($a_i[0], a_i[1], t_i, \epsilon, \epsilon/5$), where $t_i \in \{0, 1\}$. Note that $a_i[g_i]$ is the input of a process in group g_i and only processes in group $1-g_i$ write to $A[1-g_i]$. Hence $a_i[0]$ is the input of a process in group 0 and $a_i[1]$ is the input of a process in group 1. Since the inputs of all processes in the same group are within $\epsilon/5$ of one another, Lemma 1 implies that all these output points differ from one another by at most ϵ.

Now suppose that some process in group g returns its input on line 8. Since this process returns on line 8 only after writing 1 to $B[g]$, Observation 2 implies that no process writes 1 to $B[g-1]$. This implies that no processes in group $1-g$ return on line 8.

Let C be the configuration immediately following the first write to $B[g]$. Any process that returns on line 8 read $\perp$ from $A[1-g]$ on line 6 following its write to $B[g]$. Thus, the first write to $A[1-g]$ occurs after configuration C. Let C' be the configuration immediately following the first write to $A[1-g]$. Note that the first step of every process in group g is a write to $A[g]$, so $A[g] \neq \perp$ in configuration C and all subsequent configurations. Thus, each process p_j in group $1-g$ reads $a_j[g] \neq \perp$ from $A[g]$ on line 3 and reads 1 from $B[g]$ on line 12. Hence, it will call ApproxAgreeDU$(a_j[0], a_j[1], g, \epsilon, \epsilon/5)$ on line 16 with $a_j[1-g] = x_j$.

Each process p_i in group g that returns on line 10 performs its read of $A[1-g]$ on line 3 prior to C' and its read on line 6 after C'. Each process p_i in group g that returns on line 14 performs its read of $A[1-g]$ on line 3 after C'. In either case, it will call ApproxAgreeDU$(a_i[0], a_i[1], g, \epsilon, \epsilon/5)$ with $a_i[g] = x_i$.

Hence, in all calls to ApproxAgreeDU, the first argument is an input of a process in group 0, the second argument is an input of a process in group 1, and the third argument is $g \in \{0, 1\}$. Every process p_i that returns on line 10, 14, or 16 outputs $a_i[0] + g(a_i[1] - a_i[0]) = a_i[g]$, which is an input of a process in group g. If process p_i returns on line 8, it is in group g and it returns its own input. By assumption, the inputs of all processes in group g differ from one another by at most $\epsilon/5$. Hence, all outputs differ from one another by at most $\epsilon/5 < \epsilon$. ◀

Note that, if there are only two processes, then TwoGroupApproxAgree can be used to solve approximate agreement with $O(\log(S/\epsilon))$ step complexity using 1-bit multiwriter registers plus two single-writer registers to which the processes write their inputs.

3.4 Putting the Pieces Together

We can construct an algorithm for approximate agreement in $\mathbb{R}^k$, where the accuracy parameter ϵ is known by all processes and each process has no information about the inputs of the other processes. The step complexity of our algorithm depends on ($\log n$ and) the spread, the maximum distance between any two inputs, rather than the input with the largest magnitude, as in Schenk's algorithm. The idea is to use a binary tree of height $\lceil \log_2 n \rceil$, with one leaf for each process and with a separate instance of two group approximate agreement at every other node. The accuracy parameter is ϵ at the root and $\epsilon/5^d$ at internal nodes of depth d. Each process p_i traverses a path from its leaf to the root, using its input x_i as its input to the first instance of two group approximate agreement and, for each subsequent instance, using its output from the previous instance as its input. If its leaf is in the left subtree of a node, a process will be in group 0 of the instance of two group approximate agreement at the node and, if its leaf is in the right subtree, it will be in group 1.

We show that the input requirements for each instance of two group approximate agreement is satisfied.

▶ **Lemma 5.** *For $0 \leq d < \lceil \log_2 n \rceil$, for each instance of two group approximate agreement at each node of depth d and in each group, the inputs of the processes are within distance $\epsilon/5^{d+1}$ of one another.*

Proof. First, consider any node with a leaf as a child. Since the group corresponding to that child consists of only one process, the inputs of the processes in this group are all equal to one another and, hence, within distance 0 of one another.

Now consider any node which has at least one child that is not a leaf and assume the claim is true for those children. Let d be the depth of the node, so its children are at depth $d+1$. For each child that is not a leaf and for each group of that child, the inputs in the group are within distance $\epsilon/5^{d+2}$ of one another. By Lemma 4, the outputs of the instance at this child are within distance $\epsilon/5^{d+1}$ of one another. Hence the inputs to the instance at the node from the group corresponding to this child are within distance $\epsilon/5^{d+1}$ of one another. ◀

▶ **Theorem 6.** *The algorithm described above solves ϵ-unknown approximate agreement with $O(\log n(\log n + \log(S/\epsilon))$ step complexity among n processes, where S is the spread of the inputs.*

Proof. By Lemma 5, for the instance of two group approximate agreement at the root, the inputs in each group are within distance $\epsilon/5$ of one another. By Lemma 4, the outputs of the instance at the root and, hence, the algorithm are within distance ϵ of one another.

By Lemma 3, the outputs of any instance of two group approximate agreement are in the convex hull of its inputs. Thus, it follows by induction that the outputs of the algorithm are in the convex hull of $\{x_1, \ldots, x_n\}$.

Each process participates in at most one instance of two group approximate agreement at depth d for $0 \leq d < \lceil \log_2 n \rceil$. Since the inputs to this instance are all within the convex hull of $\{x_1, \ldots, x_n\}$, they are within distance S of one another. At level d, the accuracy parameter for TwoGroupApproxAgree is $\epsilon/5^d$. It involves at most one call to ApproxAgreeDU with accuracy parameter $\epsilon/5^d$, which, in turn, involves at most one call to r-agree with accuracy parameter at least $\epsilon/5^{d+1}S$. Each such call to r-agree takes $O(\log(S5^{d+1}/\epsilon) = O(d+\log(S/\epsilon))$ steps. Thus, the total step complexity is $O(\log n(\log n + \log(S/\epsilon))$. ◀

The step complexity of Schenk's algorithm for domain $\mathbb{R}$ depends only on the magnitude of the inputs, whereas the step complexity of our algorithm depends on the spread of the inputs and the number of processes. To get the best of both worlds with domain $\mathbb{R}$, one can run Schenk's algorithm and our algorithm in parallel, both with accuracy parameter $\epsilon/5$. Specifically, a separate part of shared memory is used for each algorithm and each process alternately performs steps of the two algorithms. If a process first completes Schenk's algorithm with output y, then it performs TwoGroupApproxAgree(0,y,ϵ). If it first completes our algorithm with output y, then it performs TwoGroupApproxAgree(1,y,ϵ). In either case, it returns the output it obtains from TwoGroupApproxAgree.

Note that, from the output specifications of Schenk's algorithm and our algorithm, in each group, all processes have inputs that are within distance $\epsilon/5$ of one another. Hence, from the output specifications of TwoGroupApproxAgree, all outputs will be within distance ϵ of one another. Since all three algorithms satisfy validity, the resulting algorithm also satisfies validity.

4 Lower bound on the Step Complexity as a Function of the Magnitude and the Number of Processes

In the *conflict detection* problem, each process p_i has an input $x_i \in \{1, \ldots, m\}$. If a process doesn't crash, it must output either **true** or **false**. If two processes have different input values and neither crashes, at least one of them returns **true**, indicating that there is a conflict. If all processes have the same input value, they must all output **false**, indicating no conflict. Aspnes and Ellen [3] proved that the step complexity of this problem when implemented using only registers is $\Omega\left(\min\left\{\frac{\log m}{\log\log m}, \frac{\sqrt{\log n}}{\log\log n}\right\}\right)$, where n is the number of processes.

There is a simple reduction from conflict detection to approximate agreement, where each process has an input in $\{1, \ldots, m\}$ and $\epsilon = 1/2$. Specifically, given inputs $x_i \in \{1, \ldots, m\}$, the processes perform approximate agreement to determine outputs y_i. If $y_i = x_i$, then process

p_i outputs **false** and, if $y_i \neq x_i$, then process p_i outputs **true**. If all the inputs have the same value, then, by validity, all the y_i's have this value and all processes output **false**, as required. However, if $x_i \neq x_j$, then either $y_i \neq x_i$ or $y_j \neq x_j$, since $|y_i - y_j| \leq \epsilon < 1 \leq |x_i - x_j|$. In this case, either p_i outputs **true** or p_j outputs **true**, as required. It follows that the step complexity of approximate agreement among n processes with inputs in $\{1, \ldots, m\}$ and $\epsilon = 1/2$ is $\Omega\left(\min\left\{\frac{\log m}{\log\log m}, \frac{\sqrt{\log n}}{\log\log n}\right\}\right)$.

5 Lower Bound on the Step Complexity as a Function of the Spread

Consider a wait-free algorithm for approximate agreement. For any reachable configuration C and any process p_i active in C, let $m_i(C)$ denote the value that process p_i outputs in its solo execution starting from configuration C. If p_i has already decided in configuration C, then $m_i(C)$ denotes the value that p_i decided.

▶ **Observation 7.** *If p_i is active in C, then $m_i(C) = m_i(Cp_i)$.*

▶ **Observation 8.** *If C and C' are indistinguishable to process p_i, then $m_i(C) = m_i(C')$.*

Herlihy [11] proved a lower bound for approximate agreement among 2 or more processes that communicate using single-writer registers. We present a corrected version of his proof, together with an extension to multi-writer registers. We begin with a technical lemma.

▶ **Lemma 9.** *If processes p_0 and p_1 are active in configuration C, then there exists $\sigma \in \{p_0, p_1, p_0p_1, p_1p_0\}$ such that $||m_0(C\sigma) - m_1(C\sigma)|| \geq (\sqrt{2}-1)^{|\sigma|}||m_0(C) - m_1(C)||$.*

Proof. We consider different cases depending on the operations p_0 and p_1 are poised to perform.

If p_0 is poised to perform a read, then configurations C and Cp_0 are indistinguishable to p_1, so, by Observation 8, $m_1(Cp_0) = m_1(C)$. By Observation 7, $m_0(Cp_0) = m_0(C)$. Thus $||m_0(Cp_0) - m_1(Cp_0)|| = ||m_0(C) - m_1(C)|| \geq (\sqrt{2}-1)^1||m_0(C) - m_1(C)||$.

Similarly, if p_1 is poised to perform a read, then $||m_0(Cp_1) - m_1(Cp_1)|| = ||m_0(C) - m_1(C)|| \geq (\sqrt{2}-1)^1||m_0(C) - m_1(C)||$.

If p_0 and p_1 are poised to perform writes to the same location, then Cp_0p_1 and Cp_1 are indistinguishable to process p_1, so Observation 8 implies that $m_1(Cp_0p_1) = m_1(Cp_1)$. By Observation 7, $m_1(Cp_0p_1) = m_1(Cp_0)$ and $m_1(Cp_1) = m_1(C)$. Thus $m_1(Cp_0) = m_1(C)$. However, by Observation 7, $m_0(Cp_0) = m_0(C)$. Hence, we get $||m_0(Cp_0) - m_1(Cp_0)|| = ||m_0(C) - m_1(C)|| \geq (\sqrt{2}-1)^1||m_0(C) - m_1(C)||$.

If p_0 and p_1 are poised to perform writes to different locations, then $Cp_0p_1 = Cp_1p_0$. By the triangle inequality,

$$\begin{aligned}||m_0(C) - m_1(C)|| &= ||m_0(C) - m_1(Cp_0) + m_1(Cp_0) - m_0(Cp_1) + m_0(Cp_1) - m_1(C)|| \\ &\leq ||m_0(C) - m_1(Cp_0)|| + ||m_1(Cp_0) - m_0(Cp_1)|| + ||m_0(Cp_1) - m_1(C)||.\end{aligned}$$

Since $(\sqrt{2}-1) + (\sqrt{2}-1) + (\sqrt{2}-1)^2 = 1$, it follows that either

$$\begin{aligned}||m_0(C) - m_1(Cp_0)|| &\geq (\sqrt{2}-1)||m_0(C) - m_1(C)||, \\ ||m_0(Cp_1) - m_1(C)|| &\geq (\sqrt{2}-1)||m_0(C) - m_1(C)||, \text{ or} \\ ||m_1(Cp_0) - m_0(Cp_1)|| &\geq (\sqrt{2}-1)^2||m_0(C) - m_1(C)||\end{aligned}$$

In the first case, $m_0(Cp_0) = m_0(C)$ by Observation 7, so $||m_0(Cp_0) - m_1(Cp_0)|| \geq (\sqrt{2}-1)^1||m_0(C) - m_1(C)||$.

Similarly, in the second case, $||m_0(Cp_1) - m_1(Cp_1)|| \geq (\sqrt{2}-1)^1||m_0(C) - m_1(C)||$.

In the third case, by Observation 7, $m_0(Cp_1p_0) = m_0(Cp_1)$ and $m_1(Cp_0p_1) = m_1(Cp_0)$. Since $Cp_1p_0 = Cp_0p_1$, it follows that $||m_0(Cp_1p_0) - m_1(Cp_1p_0)|| = ||m_1(Cp_0) - m_0(Cp_1)|| \geq (\sqrt{2}-1)^2||m_0(C) - m_1(C)||$. ◀

▶ **Theorem 10.** *Any approximate agreement algorithm for two or more processes and accuracy ϵ has step complexity at least $\frac{1}{2}\log_{\sqrt{2}+1}(S/\epsilon)$, where S is the spread of the inputs.*

Proof. Consider an initial configuration C_0 in which process p_0 has input x_0, process p_1 has input x_1, and $S = ||x_0 - x_1||$. Suppose an adversary schedules steps of processes p_0 and p_1 by repeatedly choosing schedules that satisfy Lemma 9 until both have output values. Let σ' be the resulting schedule, let $C' = C\sigma'$, and let $t = |\sigma|$. Then $||m_0(C') - m_1(C')|| \geq (\sqrt{2}-1)^t||m_0(C_0) - m_1(C_0)||$. To satisfy agreement, $||m_0(C') - m_1(C')|| \leq \epsilon$. To satisfy validity, $m_0(C_0) = x_0$ and $m_1(C_0) = x_1$, so $||m_0(C_0) - m_1(C_0)|| = ||x_0 - x_1|| = S$. Hence $t \geq \log_{1/(\sqrt{2}-1)}(S/\epsilon)$. This is equal to $\log_{\sqrt{2}+1}(S/\epsilon)$, since $1/(\sqrt{2}-1) = \sqrt{2}+1$. There are only two processes taking steps, so at least one of them must take at least $t/2$ steps. ◀

6 Conclusion

This paper studies wait-free multidimensional approximate agreement in the shared memory model, using only read and write operations. The step complexities of our algorithms have poly-logarithmic dependency on S/ϵ and n, where S is the maximum distance between inputs, ϵ is a parameter bounding the distance between outputs, and n is the number of processes.

There is still a gap between our upper and lower bounds, and it would be interesting to bring them closer together. In particular, it might be possible to increase the lower bound for the conflict detection problem.

Another possible avenue for future research is to explore whether our ideas can be applied in other models, in particular, to obtain approximate agreement algorithms for asynchronous message-passing systems.

References

1 Ittai Abraham, Yonatan Amit, and Danny Dolev. Optimal resilience asynchronous approximate agreement. In *Proceedings of the 8th International Conference On Principles Of Distributed Systems (OPODIS)*, pages 229–239, 2004.

2 Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873–890, 1993.

3 James Aspnes and Faith Ellen. Tight bounds for adopt-commit objects. *Theory of Computing Systems*, 55(3):451–474, 2014.

4 Hagit Attiya and Faith Ellen. *Impossibility Results for Distributed Computing*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2014.

5 Hagit Attiya, Nancy Lynch, and Nir Shavit. Are wait-free algorithms fast? *Journal of the ACM*, 41(4):725–763, 1994.

6 Hagit Attiya and Ophir Rachman. Atomic snapshots in $o(n \log n)$ operations. *SIAM Journal on Computing*, 27(2):319–340, 1998.

7 Danny Dolev, Nancy Lynch, Shlomit Pinter, Eugene Stark, and William Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):499–516, 1986.

8 Faith Ellen, Rati Gelashvili, and Leqi Zhu. Revisionist simulations: A new approach to proving space lower bounds. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 61–70, 2018.

9 Matthias Függer and Thomas Nowak. Fast multidimensional asymptotic and approximate consensus. In *Proceedings of the 32nd International Symposium on DIStributed Computing (DISC)*, pages 27:1–27:16, 2018.

10 Matthias Függer, Thomas Nowak, and Manfred Schwarz. Tight bounds for asymptotic and approximate consensus. *Journal of the ACM*, 68(6):46:1–46:35, 2021.

11 Maurice Herlihy. Impossibility results for asynchronous PRAM. In *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 327–336, 1991.

12 Gunnar Hoest and Nir Shavit. Toward a topological characterization of asynchronous complexity. *SIAM Journal on Computing*, 36(2):457–497, 2006.

13 Michiko Inoue, Toshimitsu Masuzawa, Wei Chen, and Nobuki Tokura. Linear-time snapshot using multi-writer multi-reader registers. In *Proceedings of the 8th International Workshop on Distributed Algorithms (WDAG)*, pages 130–140, 1994.

14 Hammurabi Mendes and Maurice Herlihy. Multidimensional approximate agreement in Byzantine asynchronous systems. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*, pages 391–400, 2013.

15 Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and Vijay K. Garg. Multidimensional agreement in Byzantine systems. *Distributed Computing*, 28(6):423–441, 2015.

16 Shlomo Moran. Using approximate agreement to obtain complete disagreement: The output structure of input-free asynchronous computations. In *Proceedings of the 3rd Israel Symposium on the Theory of Computing and Systems (ISTCS)*, pages 251–257, 1995.

17 Eric Schenk. Faster approximate agreement with multi-writer registers. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 714–723, 1995.

18 Nitin H. Vaidya and Vijay K. Garg. Byzantine vector consensus in complete graphs. In *Proceedings of the 32nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 65–73, 2013.