Non-Negotiating Distributed Computing

Carole Delporte-Gallet, Hugues Fauconnier, Pierre Fraigniaud, Sergio Rajsbaum, Corentin Travers

Non-Negotiating Distributed Computing

Carole Delporte-Gallet[1], Hugues Fauconnier[1], Pierre Fraigniaud[1], Sergio Rajsbaum[2], and Corentin Travers[3]

[1]IRIF, Université Paris Cité and CNRS
[2]Instituto de Matematicas, Universidad Nacional Autónoma de México
[3]LIS, Aix-Marseille Université

February 19, 2024

Abstract

A recent trend in distributed computing aims at designing models as simple as possible for capturing the inherently limited computing and communication capabilities of insects, cells, or tiny technological artefacts, yet powerful enough for solving non trivial tasks. This paper is contributing further in this field, by introducing a new mode of distributed computing, that we call *non-negotiating*. In the non-negotiating model, a process decides *a priori* what it is going to communicate to the other processes, before the computation starts, and proceeds regardless of what the process hears from others during an execution. We consider non-negotiating distributed computing in the read/write shared memory model in which processes are asynchronous and subject to crash failures. We show that non-negotiating distributed computing is *universal*, i.e., capable of solving any colorless task solvable by an unrestricted full-information algorithm in which processes can remember all their history, and send it to the other processes at any point in time. To prove this universality result, we present a non-negotiating algorithm for solving multidimensional approximate agreement, with arbitrary precision $\epsilon > 0$.

1 Introduction

1.1 Context and Objective

Standard models of distributed computing assume a set of $n \geq 2$ processes exchanging information via some communication medium. The communication media may be of very different kinds, from static to dynamic networks, and from message-passing to shared memory, to mention just a few, under various failure assumptions. Several textbooks study such models, e.g. [6, 30, 32]. There has been however much interest recently in models to study systems of limited capabilities. This is for instance the case of, e.g., sensor networks, for which space or energy considerations limit the computing power of each device. Biological systems like a flock of birds, a school of fish, or a colony of cells or insects, attracted a lot of interest from the distributed computing community recently (see, e.g., [1, 2, 14, 18]). Extremely weak models have thus been introduced, where, for instance, the processes are assumed to be finite-state automata [3], or the communication bandwidth may be drastically limited, e.g., from typically $O(\log n)$ bits in the CONGEST model [30] to $O(1)$ bits in the STONE-AGE model [12] — the BEEPING model [7] (see also [8, 9, 21]) even assumes that processes are capable to communicate only by either emitting a "beep" or remaining silent. Many of these works considered synchronous failure-free distributed computing. Others considered asynchronous models, including work assuming noisy

communication channels [13], work on self-stabilizing systems in which processes are subject to *transient* failures [20], and a few other contributions to asynchronous shared-memory computing to understand, e.g., epigenetic cell modification [33].

This paper introduces the study of a new type of limitation: the contents of all communication sent by a process is fixed before the execution starts. In all previous models we are aware of, the contents of a message sent by a process may depend on the messages it has previously received. Even in beeping models, a process decides in each round to emit a beep or remains silent based on the beeps it has heard in the previous rounds. We consider a form of *non-negotiating* distributed computing, in the sense that messages sent can only contain input values, or reflect processes' local progress, but cannot be related to processes' local histories. The non-negotiating limitation may seem too strong, especially in synchronous models, where local progress can be anyway deduced from round numbers. Is it possible to do useful non-negotiating distributed computing in an asynchronous system?

We show that the answer is *yes*, in the standard wait-free model in which n processes communicate by writing and reading in a shared memory, and processes are subject to permanent *crash* failures. The shared memory consists of an array of n single-writer/multi-reader registers (SWMR), one per process. The processes are asynchronous, and up to $n-1$ of them may crash. Algorithms in this model are *wait-free* because a process can never wait for another process to perform an action, as the latter can crash.

1.2 Non-Negotiating Distributed Computation

We propose a very weak asynchronous crash failure model. After writing its input to the shared memory, each process just repeats a loop consisting of (1) writing a private counter in its shared register, (2) reading all the counters currently present in the other registers, and (3) incrementing its private counter, until some stopping condition is fulfilled. Once the process stops, it decides an output value, and terminates. The pseudo-code of a process is presented in Algorithm 1. We call this model *non-negotiating*, because each process decides what is going to say (through

Algorithm 1 Non-Negotiating Distributed Algorithm for Input-Output Tasks

```
1: write myinput                                  ▷ Private input written (once) in shared memory
2: mycounter ← 1                                  ▷ Initialization of private counter
3: repeat
4:     write mycounter                            ▷ Private counter written in shared memory
5:     read all counters                          ▷ Counters are read sequentially, in arbitrary order
6:     mycounter ← mycounter + 1                  ▷ Private counter incremented
7: until test({counters})                         ▷ Testing the stopping condition
8: read all inputs                                ▷ Inputs are read sequentially, in arbitrary order
9: decide output({counters}, {inputs})            ▷ Computing the output
```

a sequence of write operations), before the computation starts. The i-th write operation of a process is simply i, it cannot write a value that depends on what it has read so far.

1.3 Results

In a nutshell, we show that in fact, a very large class of problems can be solved in the non-negotiating model. Moreover, we establish a universality result: any *colorless* task that is solvable wait-free by an unrestricted algorithm, is solvable wait-free by a non-negotiating algorithm. This result is established by showing how to solve multidimensional approximate agreement for an arbitrary small precision $\epsilon > 0$, using a non-negotiating algorithm, and then proving a

theorem that shows how to solve any colorless task using the multidimensional approximate agreement algorithm.

Detailed results. In *multidimensional approximate agreement*, processes start with private input values in $\mathbb{R}^d$, are required to output values in the convex hull of the inputs and are at most ϵ apart, for some parameter $\epsilon > 0$.

We start by considering *uniform* non-negotiating algorithms, that is, an even more restricted non-negotiating model, where processes execute the same number of rounds, k, fixed a priori. For $n = 2$ processes, we show that there is a uniform non-negotiating algorithm for one-dimensional approximate agreement, for any arbitrary small ϵ, using a sufficiently large value of k (Theorem 1). Always, both processes execute exactly k rounds of the algorithm, and stop.

We however show that there is no uniform non-negotiating algorithms for approximate agreement for more than two processes (Theorem 9). Remarkably, to expose its full computational power, a non-negotiating algorithm, for $n > 2$, must use, in addition to counters, the ability of a process to stop early, before executing k writes, as implied by our first main technical result: we present a non-negotiating algorithm solving multidimensional approximate agreement, for $n \geq 2$ processes (Theorem 10). It is remarkable that a process may convey information to other processes by stopping, since in a wait-free setting a process crash or stop is indistinguishable from the process just being slow.

Our second main contribution is to show that our non-negotiating algorithm for multidimensional approximate agreement can be used to solve any colorless task that is solvable by an unrestricted, full-information algorithm (Theorem 16). Our reduction to multidimensional agreement is in fact general, using our non-negotiating algorithm as a black box, and is also of independent interest.

Discussion. The class of tasks that we consider are called *colorless*, because they can be specified by sets of possible inputs to the processes, and, for each one, a set of legal sets of outputs, without referring to process IDs. Colorless tasks have been thoroughly studied, e.g., [23], as they include many of the standard tasks considered in the context of distributed computing. Note that, even for just three processes, it is undecidable whether a colorless task with finite inputs is wait-free solvable [17, 24].

The main result of the area is the Asynchronous Computability Theorem [27] characterizing asynchronous read/write shared memory task solvability in terms of topology. There is also a corresponding characterization theorem for colorless tasks [25, 26]. These and all subsequent results [23] assumed an unrestricted model: unbounded size registers and *full-information* protocols where a process remembers all its past, and includes all of it in each write operation to the shared memory [23]. We show in this paper, that for colorless tasks, a full-information protocol can be replaced by writing a counter as in a non-negotiating algorithm. However, in terms of time complexity, our approximate agreement algorithm is exponentially slow with respect to full-information algorithms [28]. We do not know if this is unavoidable, except for *uniform* algorithms, for which we prove a corresponding lower bound for the case of two processes (Theorem 8).

The approximate agreement algorithms we design assume *fixed inputs*: a process always starts with the same fixed input value. But our reduction in Section 4 shows how to solve multidimensional approximate agreement for any (finite) set of input values, because such a task is colorless [23]. Our non-negotiating multidimensional agreement algorithm is of independent interest. Indeed, since consensus is not wait-free solvable [15, 22], approximate agreement (that is, 1-dimensional approximate agreement) has been thoroughly studied since in the early days of the field [10]. The multidimensional approximate agreement version, where processes start with

values in $\mathbb{R}^d$, has also received recently much attention, e.g. [4, 19, 16]. It was considered in the context of message-passing Byzantine failures systems [29] as well as in shared memory systems with crash failures [26]. Connections between approximate agreement, distributed optimization, and machine learning have also been recently identified [11].

Organization of the paper. We present the results for two processes in Section 2. Our non-negotiating algorithm solving multidimensional agreement is described in Section 3. The fact that this algorithm can be used to solve any task that is solvable by an unrestricted algorithm is established in Section 4. Section 5 concludes the paper. Due to lack of space, some proof details have been moved to the Appendix.

2 Uniform Non-negotiated Approximate Agreement

In a *uniform* algorithm, processes always execute the same number of rounds. We present a uniform approximate agreement algorithm for two processes in Section 2.1. The proof provides intuition about the algorithm of Section 3 for any number of processes, but the intuition is only partial, since we will show that actually, for $n > 2$ processes, there is no approximate agreement uniform algorithm

We stress that both in this section and in Section 3 we design fixed inputs approximate agreement algorithms, in other words, assuming each process always starts with the same input. Approximate agreement for arbitrary (finite) inputs can be formally specified as a colorless task [23], and hence, the results in Section 4 show that they can be solved by a non-negotiating algorithm.

2.1 A Uniform Algorithm for 2-processes Approximate Agreement

We assume two processes, p_1 starts with input value 0 and process p_2 starts with input value 1. In every execution of an algorithm, each non-crashed process therefore has to decide a value in the interval $[0, 1]$, such that if only $p_i, i \in \{1, 2\}$ participates (takes steps), it decides its own input, and in any case, the decided values are at most ϵ apart.[1]

Theorem 1. *For every $\epsilon > 0$, there exists a uniform non-negotiating algorithm that solves ϵ-approximate agreement for two processes.*

Algorithm 2 is used to prove this theorem. Processes execute k rounds, where k depends on ϵ, that is, the algorithm solves $\epsilon = \frac{1}{2k+1}$-approximate agreement.

Each process $p_i, i \in \{1, 2\}$ has two local variables: a local counter c_i and an array $view_i$. The counter c_i records the local progress of p_i: it is incremented each time p_i repeats its **for** loop. The local array $view_i$ stores the successive values of the other process counter, as read by p_i. That is, for $r \in \{1, \ldots, k\}$, $view_i[r]$ is reserved to store the value returned by its r-th read operation, of the other process' register. After k iterations, process p_i decides a value y_i based on k and its $view_i$ (at line 7).

The decision of process p_i depends on the index d_i such that $view_i[d_i] + d_i \leq k$ and $view_i[d_i + 1] + d_i \geq k$. It is easy to see that such an index d_i can always be found since the function $r \to view_i[r] + r$ is increasing, and p_i performs k rounds. Moreover, in the cases when both processes terminate the algorithm, the indexes d_1 and d_2 of p_1 and p_2 are closely related, $d_1 = k - d_2$ or $d_1 = k - d_2 - 1$, as we prove (Lemma 3). Therefore, the decision values y_1 and y_2 are at most $\frac{1}{2k+1}$ apart, because $v_1 = \frac{2(k-d_1)}{2k+1}$ and $v_2 = \frac{2d_2+1}{2k+1}$, see line 7.

[1] We consider inputs 0, 1 for concreteness and following previous papers, but a similar algorithm can be designed with inputs $(1, 0)$, $(0, 1)$ as in Section 3.

Algorithm 2 Uniform non-negotiating algorithm for 2-processes $\frac{1}{2k+1}$-approximate agreement in $[0,1]$. Code for p_i, $i \in \{1,2\}$ with input $i-1$.

1: $c_i \leftarrow 0$; $view_i[r] \leftarrow 0, 1 \leq r \leq k$
2: **for** $r = 1, \ldots, k$ **do**
3: $\quad c_i \leftarrow c_i + 1$
4: $\quad \mathsf{write}(c_i)$ to $R[i]$ ▷ $R[i]$ is p_i's register
5: $\quad view_i[r] \leftarrow \mathsf{read}(R[3-i])$ ▷ read other process register
▷ store value read in the array $view_i$
6: **let** d be $\begin{cases} k \text{ if } view_i[k] = 0 \\ \text{such that } (view_i[d] \leq k-d) \wedge (view_i[d+1] \geq k-d) \text{ otherwise} \end{cases}$
7: **decide** $y_i = \begin{cases} \frac{2(k-d)}{2k+1} & \text{if } i = 1 \\ \frac{2d+1}{2k+1} & \text{if } i = 2 \end{cases}$

The validity requirement is trivially satisfied: in case only one process participates, say p_1, its array $view_1$ contains only 0's at the end of the algorithm, and the index d_1 is consequently equal to k. p_1 thus decides $\frac{2(k-d_1)}{2k+1} = 0$, as required. Similarly, in a solo execution p_2 decides 1. When both processes participate, decisions are in the range $[0,1]$.

We give a full proof of the correctness of the Algorithm below. The main technical ingredient is first to show that the arrays $view$ obtained by the processes when the algorithm terminates can be partitioned into disjoint classes $C_1, C_2, \ldots$ defined as follows.

Definition 2. *Let $view$ be an array of k integers and let $d \geq 0$ be an integer. We say that $view$ is in C_d if and only if:*

- *For $r \in \{1, \ldots, d\}$, $view[r] \in \{0, .., k-d\}$ and*
- *For $r \in \{d+1, \ldots, k\}$, $view[r] \in \{k-d, \ldots, k\}$.*

In particular, $[k, k, \ldots, k]$ belongs to the class C_0, and $[0, 0, \ldots, 0]$ belongs to the class C_k. Second, we establish that when both processes terminate the algorithm, their array $view_1$ and $view_2$ belong to two related classes:

Lemma 3. *In every execution where p_i and p_j terminate their algorithm, we have $view_i \in C_d \implies (view_j \in C_{k-d-1}) \vee (view_j \in C_{k-d})$*

This is illustrated Figure 1. In the case where both processes terminate, the views of both processes are always in "neighbour" classes. As processes chooses their output based on the index of the class their view belongs to, this ensures that outputs are at most $\epsilon = \frac{1}{2k+1}$ apart.

C_k—C_0— C_{k-1}—C_1— ..... C_2—C_{k-2}— C_1—C_{k-1}— C_0—C_k

Figure 1: An edge between C_i and C_j represents executions of the two processes ending in these classes, where red is for p_1 and blue for p_2. Thus, in one extreme p_1 decides 0 and in the other p_2 decides 1, and in adjacent vertices decisions are at most $\frac{1}{2k+1}$ apart.

An intuition behind the algorithm can be obtained by representing executions in a 2-dimensional grid, as illustrated in Figure 2. Starting from the lower left corner, an execution is depicted by moving 1 unit to the right each time p_1 executes a read or write operation, and moving up 1 unit each time p_2 executes an operation. In this example $k = 5$, and each process executes an alternating sequence of 5 write and 5 read operations. Three executions are depicted,

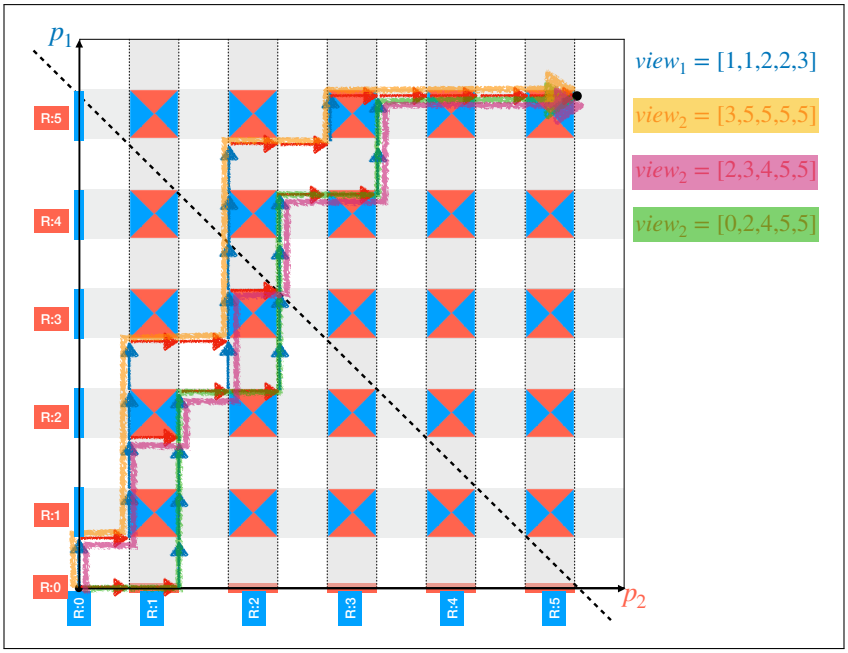

Figure 2: Three executions for $k = 5$, where p_1 has the same view $[1, 1, 2, 2, 3]$, while p_2 has different views. An horizontal arrow represents a read or write operation by process p_2 while a vertical one, an operation by process p_1. Arrows on the edge of a bi-colored read/blue cell correspond to read operations. For p_1 (respectively, for p_2), output of a read depends solely on the row index (respectively column index) of the bi-colored cell is on the edge of.

all three indistinguishable to p_1, and hence with the same $view_1$. However, p_2 distinguishes the three executions, and thus it has a different view in each one.

The diagonal $x \to k - x$ is also depicted (black dotted line.). Every path representing the execution crosses the diagonal. Notice that the last two executions (highlighted in green and purple) cross the diagonal at the same point, but not the third one. The intuition is that the decision of the process depends on when (it becomes aware from its view that) the execution has reached the diagonal. In the yellow execution, p_2 knows that the crossing point has been reached after its second read, while this happens only after its third read in the last two executions.

2.2 Correctness Proof of the 2-processes Algorithm

We first derive a few basic properties of the algorithm. In the following, $i \in \{1, 2\}$ is the index of one of the process. We denote by $j = 3 - i$ the index of the other process. When the algorithm terminates for process p_i, its local array $view_i$ contains values of the counter of process p_j, as seen by process p_i. Therefore, as each counter is incremented at most k times and initialized to 0, $0 \leq view_i[r] \leq k$ for each $r \in \{1, \ldots, k\}$. As in each iteration of its algorithm, process p_j increments its counter before writing its value to its register, the values in $view_i$ are thus non-decreasing:

Lemma 4. *If process p_i terminates its algorithm, we have*

- $\forall r, 1 \leq r \leq k : 0 \leq view_i[r] \leq k$
- $\forall r, 1 \leq r < k : view_i[r] \leq view_i[r + 1]$

Recall that class C_d contains the arrays $view$ such that:

- For $r \in \{1, \ldots, d\}$, $view[r] \in \{0, .., k-d\}$ and
- For $r \in \{d+1, \ldots, k\}$, $view[r] \in \{k-d, \ldots, k\}$.

We next check that the classes $C_1, C_2, \ldots$ are disjoint:

Lemma 5. *For every integers d, d', $d \neq d' \implies C_d \cap C_{d'} = \emptyset$*

Proof. Assume without loss of generality that $d < d'$. If $view \in C_{d'}$ then $view[d'] \in \{0, .., k-d'\}$. If $view \in C_d$ then, as $d < d'$, $view[d'] \in \{k-d, .., k\}$. As $k-d' < k-d$,$\{0, .., k-d'\} \cap \{k-d, .., k\} = \emptyset$. Hence $view$ cannot belong to both C_d and $C_{d'}$. □

Next, we prove that any array $view$ obtained when the algorithm terminates belongs to some class C_d:

Lemma 6. *If process p_i terminates its algorithm, there exists $d \in \{0, \ldots, k\}$ such that $view_i \in C_d$.*

Proof. Suppose that process p_i terminates its algorithm, and let $view_i$ be its local array at the end of the algorithm. If $view_i[1] = k$ then $view_i \in C_0$. In the following, we assume that $view_i[1] \leq k-1$.

By Lemma 4, we can identify $view_i$ with a non-decreasing continuous function $f_i : [1, k] \to [0, k]$. For example, f_i may be define as follows: on each interval $[r, r+1)$, $1 \leq r \leq k-1$, $f_i(x) = (view_i[r+1] - view_i[r])(x - r) + view_i[r]$. Let us also consider the function $g : [1, k] \to [0, k]$ that associates with each $x \in [1, k]$ $g(x) = k - x$ (Graphically, g is the diagonal line that goes through points $(1, k-1)$ and $(k, 0)$, see Figure 2).

As we assume that $view_i[1] \leq k-1$, $f_i(1) \leq g(1)$. As the codomain of f_i is $[0, k]$, $f_i(k) \geq g(k) = 0$. Since f_i is non decreasing and continuous, it thus follows that there exits an intersection point $(\alpha, k-\alpha), \alpha \in [1, k]$ in which the graphs of f_i and g intersect. For every $r \in \{1, \ldots, \lfloor \alpha \rfloor\}$, $view_i[r] \leq k - \lfloor \alpha \rfloor$, and for r from $\lfloor \alpha \rfloor + 1$ to k $view[r] \geq k - \lfloor \alpha \rfloor$. Therefore, $view_i$ is in the class $C_{\lfloor \alpha \rfloor}$. □

We are now ready to prove Lemma 3.

Lemma 3. *In every execution where p_i and p_j terminate their algorithm, we have $view_i \in C_d \implies (view_j \in C_{k-d-1}) \vee (view_j \in C_{k-d})$*

Proof. Since the value in $view_j$ are non-decreasing, it is enough to establish that $view_j[x] \leq k-x$ and $view_j[x+1] \geq k-x$ to prove that the array $view_j$ belongs to the class C_x.

Let us assume that both processes p_i and p_j terminate their algorithm, and suppose that $view_i \in C_d$ for some integer $d, 0 \leq d \leq k$. We first examine the case in which $view_i \in C_k$. This means that for every $r, 1 \leq r \leq k$, $view_i[r] = 0$. That is, process p_i writes and reads k times before the first write of p_j. Therefore, the first time process p_j reads, it sees that the counter of p_i has already reaches k. Hence, $view_j[r] = k$, for all $r, 1 \leq r \leq k$. It thus follows that $view_j \in C_0$, as desired.

We now assume that $view_i \in C_d$ for some $d, 0 \leq d < k$. For $\ell \in \{1, 2\}$ and $r, 1 \leq r \leq k$, let W_ℓ^r and R_ℓ^r denote respectively the r-th write and the r-th read by process p_ℓ.

As $view_i \in C_d$, the read R_i^d returns a value $\leq k-d$ and the next read R_i^{d+1} returns a value $\geq k-d$. This implies that process p_j cannot write a value larger than $k-d$ too early, that is before R_i^d occurs. Similarly, p_j cannot write values smaller than $k-d$ after R_i^{d+1} occurs. As writing alternates with reading, we have the two following claims :

Claim C1 R_j^{k-d+1} occurs after W_i^d

Claim C2 R_j^{k-d-1} occurs before W_i^{d+2}

Proof of Claim C1 Assume for contradiction that the $(k-d+1)$-th read by process p_j precedes the d-th write by process p_j. Therefore p_j has written $k-d+1$ to its register before the d-th read by p_i (this is because W_j^{k-d+1} precedes R_j^{k-d+1}, and p_j writes $k-d+1$ the $(k-d+1)$-th time it writes). Hence, $view_i[d] \geq k-d+1$, contradicting the fact that $view_i \in C_d$.

Proof of Claim C2 Assume for contradiction that R_j^{k-d-1} follows W_i^{d+2}. This means that when W_i^{d+2} occurs, the largest value written in the register of p_j is smaller than or equal to $k-d-1$. In particular, as R_i^{d+1} precedes W_i^{d+2}, the $(d+1)th$ read by process p_i returns a value smaller than or equal to $k-d-1$. Hence, $view_i[d+1] \leq k-d-1$, contradicting the fact $view_i \in C_d$.

We deduce from the two claims that $view_j$ belongs to C_{k-d} or C_{k-d-1}. Assume that $view_j \notin C_{k-d}$. By Claim *C2*, $view_j[k-d-1] \leq d+1 = k-(k-d-1)$. By Claim *C1*, $view_j[k-d+1] \geq d = k-(k-d)$. As we assume that $view_j \notin C_{k-d}$, it must be the case that $view_j[k-d] \geq d+1 = k-(k-d-1)$. It thus follows that $view_j \in C_{k-d-1}$. □

Lemma 7. *Algorithm 2 implements $\frac{1}{2k+1}$-approximate agreement with fixed inputs for two processes.*

Proof. Let $\ell \in \{1,2\}$ and let us assume that only process p_ℓ participates. Its array $view_\ell$ is thus equal to $[0,\ldots,0]$ at the end of the algorithm. By definition 2, $[0,\ldots,0] \in C_k$. By the code of Algorithm 2, p_ℓ decides 0 if $\ell = 1$ and 1 if $\ell = 2$, as required (recall that the input of each process is fixed: 0 for process p_1 and 1 for process p_2.).

Let us now consider the case in which both processes participate. By Lemma 6, if process p_ℓ, $\ell \in \{1,2\}$ terminates its algorithm, $view_\ell \in C_d$ for some $d \in \{0,\ldots,k\}$. From Lemma 5, this d is unique. If follows from the code (line 7) and the fact that $0 \leq d \leq k$ that decided values are in the range $[0,1]$.

Suppose that both processes decide and let d_1 and d_2 be such that their array $view_1$ and $view_2$ are in the classes C_{d_1} and C_{d_2} respectively at the end of their algorithm. By the code (line 7), process p_1 decides $v_1 = \frac{2(k-d_1)}{2k+1}$ and process p_2 decides $v_2 = \frac{2d_2+1}{2k+1}$. By Lemma 3, $d_2 = k-d_1-1$ or $d_2 = k-d_1$. Therefore,

$$v_1 - v_2 = \frac{2k-2d_1-2d_2-1}{2k+1} = \begin{cases} \frac{1}{2k+1} \text{ if } d_2 = k-d_1-1 \\ \frac{-1}{2k+1} \text{ if } d_2 = k-d_1 \end{cases}$$

Hence, if processes p_1 and p_2 terminate their algorithm, their decision value differs by at most $\frac{1}{2k+1}$. □

Theorem 1 is an immediate consequence of Lemma 7.

2.3 Inherent Slowness of Uniform Non-negotiating computing

We show that our uniform two-processes algorithm is essentially optimal with regards to the number of rounds k. Thus, although it is exponentially slower than unrestricted algorithms such as [28], this is unavoidable. Indeed, we establish that $\Omega(\frac{1}{k})$ is a lower bound on the agreement parameter ϵ of any uniform non-negotiating algorithm that stops after k rounds, for any k:

Theorem 8. *For any $\epsilon < \frac{1}{4k-1}$, there is no uniform k-non negotiating algorithm for two processes that implements ϵ-agreement.*

The main idea is to consider the subset of executions of a uniform k-non negotiating algorithm that have an immediate snapshot schedule, following [5]. Such an execution is defined by a sequence of concurrency classes, $c_1, c_2, \ldots$, where each c_i consists of a non-empty subset of $\{p_1, p_2\}$. In the corresponding execution, processes in c_i write their counters (in an arbitrary order), and then they read each other counters (in an arbitrary order). A sequence of concurrency classes $C = c_1, c_2, \ldots, c_\ell$ is *complete* if each process appears in exactly k concurrency classes, i.e., it is an executions of a *uniform* k-non negotiating algorithm. Notice that the number ℓ of concurrency classes satisfies $k \leq \ell \leq 2k$.

Some immediate snapshot executions are represented in Figure 3, where the horizontal axis corresponds to the operations executed by p_1 and the vertical axis by p_2. Thus, a path on the grid defines an interleaving of the operations of the two processes. Arrows represent concurrency classes. Table (a) represents the fully synchronous execution of 4 concurrency classes, where every concurrency class c_i is equal to $\{p_1, p_2\}$, while Table (h) represents a fully sequential execution of 8 concurrency classes, where first p_1 executes and then p_2, so the first 4 concurrency classes c_i are equal to $\{p_1\}$, and the other 4 concurrency classes are equal to $\{p_2\}$. Notice that a full concurrency class $\{p_1, p_2\}$ is represented as a diagonal arrow, for the two write operations, and for the two read operations, since both commute: no processes distinguishes in which order the operations were performed. But in brief, the idea is to count the number of executions from

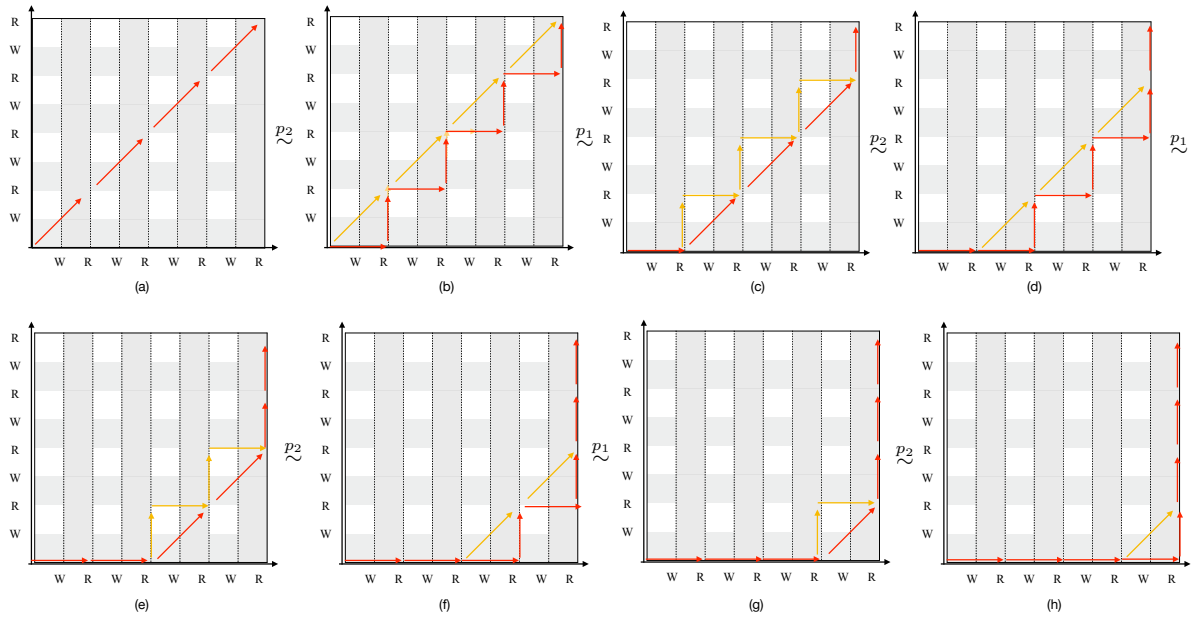

Figure 3: Some immediate snapshot executions of a uniform k-non negotiating algorithm, $k = 4$, and their indistinguishability relations.

the fully sequential to the fully concurrent. In more detail, there is a path of $4k - 1$ executions (vertices), with C_1 as the central vertex, and whose endpoints are the fully sequential executions, C', C'', and each two consecutive executions are indistinguishable to one process. This claim implies that the best ϵ-agreement that can be achieved is $\frac{1}{4k-1}$. This is because in each two consecutive executions one process decides the same value.

Recall that a *uniform k-non negotiating algorithm* is a non-negotiating algorithm in which processes perform k iterations of the repeat loop in every execution, while in a *k-non negotiating algorithm* processes performs *at most* k iterations of the repeat loop in every execution. In such an algorithm, processes are thus allowed to stop early, as a function of their views of the other

processes' counters. We next present the proof of Theorem 8 (restated below).

Theorem 8. *For any $\epsilon < \frac{1}{4k-1}$, there is no uniform k-non negotiating algorithm for two processes that implements ϵ-agreement.*

Proof. For the proof it suffices to consider the subset of executions of an uniform k-non negotiating algorithm that have an immediate snapshot schedule, following [5]. Such an execution is defined by a sequence of concurrency classes, $c_1, c_2, \ldots$, where each c_i consists of a non-empty subset of $\{p_1, p_2\}$. In the corresponding execution, processes in c_i write their counters (in an arbitrary order), and then they read each other counters (in an arbitrary order). A sequence of concurrency classes $C = c_1, c_2, \ldots, c_\ell$ is *complete* if each process appears in exactly k concurrency classes, i.e., it is an execution of a *uniform* k-non negotiating algorithm. Notice that the number of concurrency classes satisfies $k \leq \ell \leq 2k$.

Some immediate snapshot executions are represented in Figure 4, where the horizontal axis corresponds to the operations executed by p_1 and the vertical axis by p_2. Thus, a path on the grid defines an interleaving of the operations of the two processes. Arrows represent concurrency classes. Table (a) represents the fully synchronous execution of 4 concurrency classes, where every concurrency class c_i is equal to $\{p_1, p_2\}$, while Table (h) represents a fully sequential execution of 8 concurrency classes, where first p_1 executes and then p_2, so the first 4 concurrency classes c_i are equal to $\{p_1\}$, and the other 4 concurrency classes are equal to $\{p_2\}$. Notice that a full concurrency class $\{p_1, p_2\}$ is represented as a diagonal arrow, for the two write operations, and for the two read operations, since both commute: no processes distinguishes in which order the operations were performed. On the other hand, we have the following observation.

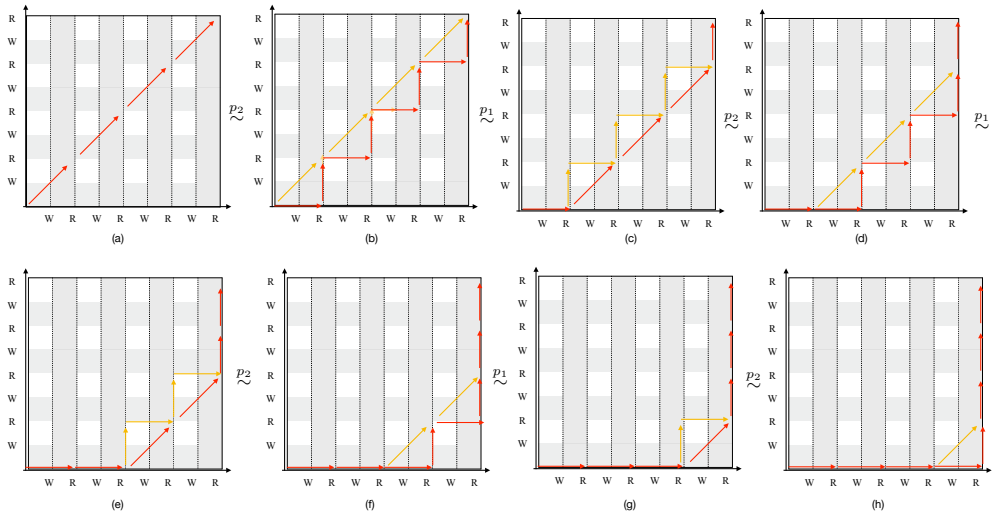

Figure 4: Some immediate snapshot executions of a uniform k-non negotiating algorithm, $k = 4$, and their indistinguishability relations.

Consider two executions $C = c_1, c_2, \ldots, c_\ell$ and $C' = c'_1, c'_2, \ldots, c'_{\ell'}$. We write $C \overset{p_i}{\sim} C'$ if p_i does not distinguish the two executions, i.e., it reads the same sequence of counters in C and in C'.

Observation: **(i)** Any concurrency class $c_i = \{p_1, p_2\}$ of C, can be split into two consecutive concurrency classes $\{p_1\}\{p_2\}$ to obtain and execution C' indistinguishable to p_2. **(ii)** Similarly, any two consecutive concurrency classes $\{p_1\}\{p_2\}$ of C can be joined into one $\{p_1, p_2\}$ and obtain an execution C' indistinguishable to p_2.

The observation can be applied in "parallel", as illustrated in Figure 4. For example, Table (a) and Table (b), represent executions indistinguishable to p_2, obtained by splitting all concurrency classes in Table (a). Similarly, Table (c) is obtained from Table (b) by joining concurrency classes, maintaining indistinguishability with respect to p_1.

Now, consider the concurrent execution $C_1 = c_1, c_2, \ldots, c_k$, and the two sequential executions, $C' = c'_1, c'_2, \ldots, c'_{2k}$, and $C'' = c''_1, c''_2, \ldots, c''_{2k}$, where in C', first p_1 runs solo and then p_2 runs solo, and in C'' the opposite. Namely, for $1 \leq i \leq k$, we have $c'_i = \{p_1\}$, $c'_{k+i} = \{p_2\}$, and $c''_i = \{p_2\}$ and $c''_{k+i} = \{p_1\}$.

Claim: There is a path of $4k-1$ executions (vertices), with C_1 as the central vertex, and whose endpoints are the fully sequential executions, C', C'', and each two consecutive executions are indistinguishable to one process.

This Claim implies that the best ϵ-agreement that can be achieved is $\frac{1}{4k-1}$. This is because each two consecutive executions one process decides the same value.

We verify the above claim, following the example in the figure. Let us denote by α_m a sequence of m concurrency classes $\{p_1, p_2\}$, β^i_m a sequence of m concurrency classes $\{p_i\}$, by γ^{ij}_m a sequence of m concurrency classes $\{p_i\}, \{, p_j\}$. Thus, $C_1 = \alpha_k$.

Applying repeatedly the previous observation, we have the following sequence of executions illustrated in Figure 4,

$$C_1 \overset{p_2}{\sim} C_2 \overset{p_1}{\sim} C_3 \overset{p_2}{\sim} \cdots \overset{p_2}{\sim} C_{2k} (= C')$$

and

$$C_1 = C'_1 \overset{p_1}{\sim} C'_2 \overset{p_2}{\sim} C'_3 \overset{p_1}{\sim} \cdots \overset{p_1}{\sim} C'_{2k} (= C'')$$

where for $1 \leq i \leq k$, C_{2i-1} is of the form $\beta^1_{i-1}\alpha_{k-i+1}\beta^2_{i-1}$, and C'_{2i-1} is of the form $\beta^2_{i-1}\alpha_{k-i+1}\beta^1_{i-1}$. □

3 Non-Negotiated Multidimensional Approximate Agreement

In this section, we show how to solve multidimensional approximate agreement in the non-negotiating model. As in the previous section, we concentrate on a *fixed inputs* version of *multidimensional approximate agreement* in which each process starts with a fixed input value. It will be used in Section 4 to solve *any* colorless task. Let $\epsilon > 0$. Each process $p_i, i \in \{1, \ldots, n\}$, $n \geq 2$, starts with input $\mathbf{x}_i = (0, \ldots, 0, 1, 0, \ldots, 0) \in \{0,1\}^n$ where the unique 1 stands at the i-th coordinate. Each participating process p_i must output a vector $\mathbf{y}_i \in \mathbb{Q}^n$ such that, if $I \subseteq \{1, \ldots, n\}$ denotes the set of participating processes, then (1) for every non-faulty process $p_i \in I$, $\mathbf{y}_i \in \mathsf{Hull}(\{\mathbf{x}_i \mid i \in I\})$, and (2) for every two non-faulty processes $p_i, p_j \in I$, $||\mathbf{y}_i - \mathbf{y}_j||_2 \leq \epsilon$. Here, for a set S of points in $\{0,1\}^n$, $\mathsf{Hull}(S)$ denotes the convex hull of S. Note that since, for every i, process p_i always starts with input $\mathbf{x}_i$, it does not need to write $\mathbf{x}_i$ in memory.

We first show that actually no uniform algorithm can solve multidimensional approximate agreement (for arbitrarily small ϵ).

Theorem 9. *Let $n \geq 3$. For $\epsilon < \sqrt{\frac{n-2}{n-1}}$, there is no* uniform *non-negotiating algorithm that implements multidimensional approximate agreement for n processes.*

Proof. Let $\mathcal{A}$ be a n-processes uniform non-negotiating algorithm that implements multidimensional ϵ-agreement for some $\epsilon, 0 \leq \epsilon < 1$. Recall that in our fixed inputs version, the input of each process p_i is the point $\mathbf{u}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ whose all coordinates are 0, except the i-th which is 1.

In a uniform non-negotiating algorithm, in every execution, every process performs k iterations of the repeat loop before deciding (unless it fails). The constant k may depend on ϵ, but it is independent of the execution.

The proof is based an indistinguishability argument. For $i, 1 \leq i \leq n-1$, let e_i be an execution such that:

- Process p_i performs first its k iterations and decides,
- Then processes $p_j, j \in \{1, \ldots, n-1\} \setminus \{i\}$ perform their k iterations of the repeat loop. The order in which these processes take steps does not matter. For example, they may perform their k iterations sequentially in increasing index order.
- Finally, process p_n performs its k iterations of the for loop of the algorithm.

Let us first observe that for each $i, 1 \leq i \leq n-1$, process p_i cannot distinguish e_i from an execution in which it is the only participating process. It thus decides $\mathbf{u}_i$ in execution e_i.

Second, note that executions $e_1, \ldots, e_{n-1}$ are indistinguishable for process p_n. Indeed, it each of them, the state of the shared memory is $[k, \ldots, k, 0]$ when p_n starts executing its algorithm as every process has written the final value, k, of its counter before p_n has taken even one step. Hence, in each execution $e_i, 1 \leq i \leq n-1$, the successive views of the counters for process p_n is $[k, \ldots, k, 1], \ldots, [k, \ldots, k, k]$. p_n thus decides the same point $\mathbf{y}$ in $e_1, \ldots, e_{n-1}$.

By the first observation, it must be the case that $||\mathbf{y} - \mathbf{u}_i||_2 \leq \epsilon$ for all $i, 1 \leq i \leq n-1$. The coordinates of the point $\mathbf{c}$ closest to $\mathbf{u}_1, \ldots, \mathbf{u}_{n-1}$ (that is, the centroid of $\mathbf{u}_1, \ldots, \mathbf{u}_{n-1}$) are $(\frac{1}{n-1}, \ldots, \frac{1}{n-1}, 0)$. As $||\mathbf{c} - \mathbf{u}_i||_2 = \sqrt{\frac{n-2}{n-1}}$ for every $i, 1 \leq i \leq n-1$, it follows that $\epsilon \geq \sqrt{\frac{n-2}{n-1}}$. □

We now present our general multidimensional approximate agreement algorithm, which in light of the previous impossibility, is not uniform.

Theorem 10. *For every $n \geq 2$, and for every $\epsilon > 0$, Algorithm 3 with $k = \frac{3n}{\epsilon}$ solves multidimensional ϵ-approximate agreement for n processes.*

Algorithm 3 Non-negotiating Algorithm for multidimensional approximate agreement. Code for process $p_i, i \in \{1, \ldots, n\}$, starting with input $\mathbf{x}_i$. R is a shared array of n integers initialized to 0.

```
1: c_i ← 0                                          ▷ Counter of p_i
2: view_i ← [0, ..., 0]                             ▷ view_i[j] contain last read value of p_j's counter
3: repeat
4:     c_i ← c_i + 1
5:     write(c_i) to R[i]                           ▷ R[i] is the shared register of p_i
6:     view_i ← collect(R)                          ▷ instruction collect reads all registers in arbitrary order
7: until Σ_{j=1}^{n} view_i[j] ≥ k                  ▷ stop when the sum of counters reaches threshold k
8: decide y_i = (Σ_{j=1}^{n} (view_i[j] · x_j)) / (Σ_{j=1}^{n} view_i[j])
```

The rest of the section is dedicated to the proof of Theorem 10. In Algorithm 3, the accuracy ϵ of the agreement is controlled through parameter $k = O(\frac{n}{\epsilon})$. A process stops incrementing its counter when it observes that the sum of the counters is larger than the threshold k (line 7). In other words, each process looks for the first iteration d at which the sum of the counters of the processes is at least k. In each iteration of the repeat-loop, the value of the counters, as observed by process p_i, are stored in the local variable $view_i$, and are interpreted as the coordinates of a point $\mathbf{d}_i \in \mathbb{R}^n$ where

$$\mathbf{d}_i = \sum_{j=1}^{n} (view_i[j] \cdot \mathbf{x}_j).$$

The algorithm stops at process p_i when $\mathbf{d}_i$ is far enough from the origin $(0, \ldots, 0)$, that is, when it is on the other side of the hyperplane H_k with respect to the origin — see Figure 5. H_k is

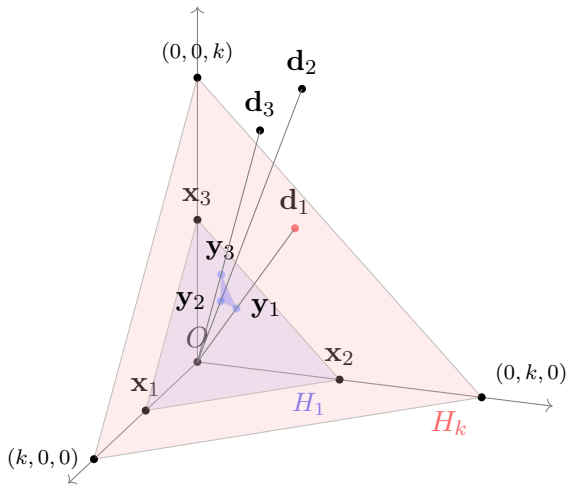

Figure 5: Points $\mathbf{d}_i$, $i \in \{1, 2, 3\}$, and outputs $\mathbf{y}_i$ for $n = 3$.

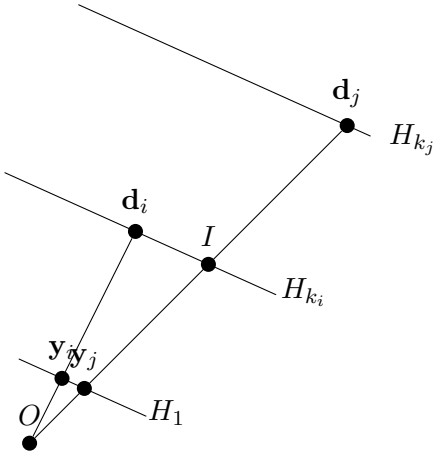

Figure 6: The points used in the proof of Lemma 15

the hyperplane that contains the n points $(k, 0, \ldots, 0), \ldots, (0, \ldots, 0, k)$. The coordinates of $\mathbf{d}_i$ are then scaled down to produce a decision $\mathbf{y}_i$ that belongs to the hyperplane H_1 (line 8). As we shall prove in Lemma 12, scaling down the points $\mathbf{d}_i$ ensures validity as the resulting points $\mathbf{y}_i$ are in the convex hull of the unit vectors corresponding to the participating processes. For agreement, we shall prove in Lemma 15 that the decisions $\mathbf{y}_i, i \in I$, all lie in a same L_2-ball of diameter $O(\frac{n}{k})$.

We first show that Algorithm 3 terminates.

Lemma 11. *In every execution, every process decides after a finite number of iterations.*

Proof. Let p_i be a process. Suppose for contradiction that there is an infinite execution in which p_i does not fail, and does not decide. This means that p_i performs infinitely many iterations of the repeat-loop. In particular, in iteration k, the value of the counter of process p_i is k, which is the value that p_i writes to shared memory in this iteration. Hence, the view it obtains after reading the memory in this iteration is such that $view_i[i] = k$. Since the initial value of each register is 0 and each process writes only positive values to its register, we get that $\sum_{1 \leq j \leq n} view_i[j] \geq k$. Therefore, process p_i exits the repeat-loop, and, as it does not fail, decides at line 8. □

Next, we show that the validity condition of multidimensional ϵ-agreement is satisfied.

Lemma 12. *In every execution, any decision is in the convex hull of the inputs of the participating processes.*

Proof. Let e be an execution of Algorithm 3, let P be the set of participating processes in e, and let $p_i \in P$ be a process that decides in e. We denote by $\mathbf{y}_i$ its decision. It is required that any decision $\mathbf{y}_i$ belongs to the convex hull of the inputs of the participating processes, that is, $\mathbf{y} = \sum_{j \in P} \lambda_j \mathbf{x}_j$ where, for every $j \in P$, $\lambda_j \geq 0$ and $\sum_{j \in P} \lambda_j = 1$. Let $view_i$ be the result of the last collect by process p_i before it decides. Note that for each $p_j \notin P$, $view_i[j] = 0$ as a non-participating process never writes to shared memory, and all registers are initialized to 0. By Line 8, the coordinates of the output decision $\mathbf{y}_i$ are $\frac{1}{\sum_j view_i[j]}(view_i[1], \ldots, view_i[n])$. Therefore, the coordinates are all positive, and their sum is 1. Since, in addition, $view_i[j] = 0$ for every $j \notin P$, it follows that $\mathbf{y}_i \in \mathsf{Hull}(\{\mathbf{x}_j : j \in P\})$. □

We now establish a couple of technical lemmas that will be used for proving that the outputs are close to each other.

Lemma 13. *Let e be an execution, and let $i \in \{1, \ldots, n\}$ such that process p_i is correct in e. Let $view_i$ be the last view of p_i before p_i decides. Then*

$$k \leq \sum_{1 \leq j \leq n} view_i[j] \leq k + n - 1.$$

Proof. As $view_i$ is obtained the last times process p_i reads the shared memory, we have $k \leq \sum_{1 \leq j \leq n} view[j]$ by the stopping condition in line 7. Assume for contradiction that $\sum_{1 \leq j \leq n} view_i[j] \geq n + k$. For every integer ℓ, $1 \leq \ell \leq n + k$, let W_ℓ denote the ℓth write to the shared memory. As registers are initialized to 0, and since each time a process writes to shared memory it increments its register by 1, any read of the memory that terminates before the write W_ℓ returns an array whose sum of its components is less than ℓ. Similarly, if an array v is obtained by a read of the memory that starts after W_ℓ, then $\ell \leq \sum_{1 \leq j \leq n} v[j]$. As process p_i obtains the array $view_i$ with $\sum_{1 \leq j \leq n} view_i[j] \geq n + k$, the execution must contain at least $n + k$ writes. In particular the $n + 1$ writes $W_k, \ldots, W_{k+n}$ must occur in e. At least two of them, say W_ℓ and W'_ℓ, with $k \leq \ell < \ell' \leq k + n$, must be performed by a single process, denoted by p_j. After performing the write operation W_ℓ, p_j reads the memory. The sum of the counters in the array that it obtains by reading is at least $\ell \geq k$. It follows that p_j exits its repeat-loop, and consequently never performs $W_{\ell'}$, a contradiction. □

The previous lemma establishes a bound on the sum of the components of the final views of the processes. This sum cannot be too far away from the threshold value k. In the next lemma, we examine each component individually, and we show that each component j cannot be to far from some value c_j.

Lemma 14. *Let e be a finite execution in which at least one process decides. For each process p_j, $j \in \{1, \ldots, n\}$, let c_j be the last value of its counter as written to its register, with $c_j = 0$ if process p_j does not participate in e. For every process p_i that decides there exists n non-negative integers $\delta_i^1, \ldots, \delta_i^n$ such that $\sum_{\ell=1}^n \delta_i^\ell \leq n - 1$, and $\mathbf{d}_i = \sum_{\ell=1}^n (c_\ell - \delta_i^\ell) \cdot \mathbf{x}_\ell$.*

Proof. As in the proof of Lemma 13, let W_ℓ denote the ℓth write to the shared memory in e. Since the registers are initialized to 0, and since each write operation increments a register by 1, we have $\sum_{1 \leq j \leq n} R[j] = \ell - 1$ immediately before W_ℓ occurs, and $\sum_{1 \leq j \leq n} R[j] = \ell$ immediately after.

We say that a process is *slow* if its last write precedes W_k, and that a process is *fast* if its last write follows W_k. Let S and F be the sets of indexes of slow and fast processes, respectively. In addition, let p_t be the process that performs the write W_k. Let p_i be a process that decides, and, let $view_i$ be the result of its last collect operation. We now examine how each individual component $view_i[j]$ compare to the last value c_j written by p_j in its register. We distinguish the two cases $i \in F \cup \{t\}$ and $i \in S$.

If $i \in F \cup \{t\}$, then p_i is fast, or $i = t$. In this case, the last collect performed by p_i starts after W_k. Therefore, for each $\ell \in S \cup \{t\}$, $view_i[\ell] = c_\ell$. For $\ell \in F$, note that any write performed by a process p after W_k is the last write of this process. Indeed, the collect that follows that write returns an array whose sum is at least k, since $\sum_{1 \leq j \leq n} R[j] \geq k$ after W_k. Therefore, since the last time process p_i reads $R[\ell]$ occurs after W_k, that read either returns the last or the next-to-last value written by p_ℓ to $R[\ell]$. Hence $view_i[\ell] = c_\ell - \delta_i^\ell$ where $\delta_i^\ell \in \{0, 1\}$. To summarize, for every $\ell \in \{1, \ldots, n\}$, we have $view_i[\ell] = c_\ell - \delta_i^\ell$ where $\delta_i^\ell = 0$ if $\ell \in S \cup \{t\}$, and $\delta_i^\ell \in \{0, 1\}$ if $\ell \in F$. Since there are at most $n - 1$ fast processes, we get that $\sum_{1 \leq \ell \leq n} \delta_i^\ell \leq n - 1$.

If $i \in S$ then the last collect of p_i starts before W_k. The value read from register $R[\ell]$ by p_i is $c_\ell - \delta_i^\ell$, where $\delta_i^\ell \geq 0$. On one hand, process p_i decides, and thus the sum of the components of $view_i$ must be larger that k. That is,

$$\sum_{1 \leq \ell \leq n} c_\ell - \delta_i^\ell \geq k. \tag{1}$$

On another hand, k can be expressed in terms of the last value of the counters of the processes:

$$k = \sum_{j \in R} (c_j - 1) + \sum_{j \in S} c_j + c_t. \tag{2}$$

Indeed, we know that $\sum_{1 \leq j \leq n} R[j] = k$ immediately after W_k. The value stored in $R[j]$ is then c_j if p_j is a slow process or if $p_j = p_t$, and it is $c_j - 1$ if p_j is a fast process. This is because, as we have seen before, a fast process performs exactly one write after W_k. By combining Eq. (1) and (2), we get $\sum_{1 \leq \ell \leq n} \delta_i^\ell \leq |F|$, from which we conclude that $\sum_{1 \leq \ell \leq n} \delta_i^\ell \leq n - 1$ as there are at most $n - 1$ fast processes.

It follows that, in each case, there exists n non-negative integers $\delta_i^1, \ldots, \delta_i^n$ such that

$$\mathbf{d}_i = \sum_{1 \leq \ell \leq n} view_i[\ell] \cdot \mathbf{x}_\ell = \sum_{1 \leq \ell \leq n} (c_\ell - \delta_i^\ell) \cdot \mathbf{x}_\ell$$

with $\sum_{1 \leq \ell \leq n} \delta_i^\ell \leq n - 1$, as desired. □

We now have all the ingredients to show that multidimensional ϵ-agreement is solved by Algorithm 3.

Lemma 15. *Let us assume that, in some execution of Algorithm 3, p_i and p_j decide $\mathbf{y}_i$ and $\mathbf{y}_j$, respectively. Then $||\mathbf{y}_i - \mathbf{y}_j||_2 \leq \frac{3(n-1)}{k}$.*

Proof. We denote by $view_i$ and $view_j$ the last counters collected on the memory obtained by p_i and p_j, respectively. If $view_i = view_j$, then $\mathbf{y}_i = \mathbf{y}_j$, and the lemma follows. In the following, we thus suppose that $view_i \neq view_j$. Let us denote by k_i (respectively, k_j) the sum of the components of $view_i$ (respectively, $view_j$). Without loss of generality, we assume that $k_i \leq k_j$. For every integer $\ell > 0$, let H_ℓ be the hyperplane that contains the n points $(\ell, 0, \ldots, 0), \ldots, (0, \ldots, 0, \ell)$. That is, $H_\ell = \{\mathbf{v} = (v_1, \ldots, v_n) \mid \sum_{1 \leq \lambda \leq n} v_\lambda = \ell\}$. Note that $\mathbf{d}_i \in H_{k_i}$ as it is the point whose coordinates are $(view_i[1], \ldots, view_i[n])$.

The line $(O\mathbf{d}_i)$, where $O = (0, \ldots, 0)$ is the origin, intersects H_k in a single point that we denote by I — See Figure 6. Indeed, $\mathbf{d}_j$ is either in H_{k_i}, in which case $I = \mathbf{d}_j$, or on the opposite side of H_k. By line 8 in Algorithm 3, $\mathbf{y}_i$ stands on the line $(O\mathbf{d}_i)$. Similarly, $\mathbf{y}_i$ stands on the line $(O\mathbf{d}_j)$, and, by definition, I also stands on this line. Therefore, points O, $\mathbf{d}_i$, $\mathbf{d}_j$, $\mathbf{y}_i$, $\mathbf{y}_j$, and I are co-planar. To bound the distance between $\mathbf{y}_i$ and $\mathbf{y}_j$, we are going to use the Intercept Theorem on the triangles $(O\mathbf{d}_i I)$ and $(O\mathbf{y}_i\mathbf{y}_j)$.

We first show that we can bound the L_1-norm $||I - \mathbf{d}_i||_1$ from above by a quantity that does not depend on k. Observe that

$$I = \frac{k_i}{k_j} \mathbf{d}_j$$

Indeed, $\mathbf{d}_j = (view_j[1], \ldots, view_j[n])$ with $\sum_{1 \leq \ell \leq n} view_j[\ell] = k_j$. Hence $\frac{k_i}{k_j}\mathbf{d}_j$ is on the hyperplane H_{k_i} and also a point on the line passing through O and $\mathbf{d}_j$. We use the expression of $\mathbf{d}_i$ and $\mathbf{d}_j$ given by Lemma 14. Recall that c_ℓ is the last value written by process p_ℓ to its register, and $\delta_t^1, \ldots, \delta_t^n, t \in \{i, j\}$ are non-negative integers whose sum is at most $n - 1$. We have

$$\mathbf{d}_i = \sum_{1 \leq \ell \leq n} (c_\ell - \delta_i^\ell)\mathbf{x}_\ell, \text{ and } \mathbf{d}_j = \sum_{1 \leq \ell \leq n} (c_\ell - \delta_j^\ell)\mathbf{x}_\ell,$$

from which we deduce that

$$\mathbf{d}_j = \mathbf{d}_i + \sum_{1\leq \ell \leq n} (\delta_i^\ell - \delta_j^\ell)\mathbf{x}_\ell$$

We next use this equation to bound $||\mathbf{d}_i - I||_1$, as follows.

$$\begin{aligned}
||\mathbf{d}_i - I||_1 &= ||\mathbf{d}_i - \frac{k_i}{k_j}\mathbf{d}_j||_1 \\
&= ||\mathbf{d}_i - \frac{k_i}{k_j}(\mathbf{d}_i + \sum_{1\leq \ell \leq n} (\delta_i^\ell - \delta_j^\ell)\mathbf{x}_\ell)||_1 \\
&= ||(1 - \frac{k_i}{k_j})\mathbf{d}_i - \frac{k_i}{k_j}\sum_{1\leq \ell \leq n} (\delta_i^\ell - \delta_j^\ell)\mathbf{x}_\ell||_1 \\
&\leq (1 - \frac{k_i}{k_j})||\mathbf{d}_i||_1 + \frac{k_i}{k_j}||\sum_{1\leq \ell \leq n} (\delta_i^\ell - \delta_j^\ell)\mathbf{x}_\ell||_1
\end{aligned}$$

Note that

$$||\mathbf{d}_i||_1 = \sum_{1\leq \ell \leq n} |view_i[\ell]| = \sum_{1\leq \ell \leq n} view_i[\ell] = k_i.$$

Also, since, first, $\sum_{1\leq \ell \leq n} \delta_i^\ell \leq n-1$, second, $\sum_{1\leq \ell \leq n} \delta_j^\ell \leq n-1$, and, third, all terms $\delta_i^\ell, \delta_j^\ell$ are non-negative, we have $||\sum_{1\leq \ell \leq n}(\delta_i^\ell - \delta_j^\ell)\mathbf{x}_\ell||_1 \leq 2(n-1)$. Therefore,

$$||\mathbf{d}_i - I||_1 \leq k_i(1 - \frac{k_i}{k_j}) + 2(n-1)\frac{k_i}{k_j} = \frac{(k_j - k_i + 2(n-1))k_i}{k_j}. \tag{3}$$

Let $k_j = \Delta + k_i$. Since k_i and k_j are the sums of the components of $view_i$ and $view_j$ obtained by p_i and p_j, respectively, we get that $\Delta \leq n-1$ by Lemma 13. In addition, $0 \leq \Delta$ as we assume $k_i \leq k_j$. Equation (3) can thus be rewritten as follows:

$$\begin{aligned}
||\mathbf{d}_i - I||_1 &\leq \frac{(k_i + \Delta - k_i + 2(n-1))k_i}{k_i + \Delta} \\
&\leq \frac{(\Delta + 2(n-1))}{1 + \frac{\Delta}{k_i}} \leq 3(n-1)
\end{aligned} \tag{4}$$

Due to the relationship between L_2-norm and L_1-norm, we have

$$||\mathbf{d}_i - I||_2 \leq ||\mathbf{d}_i - I||_1 \leq 3(n-1)$$

Finally, we use the Intercept Theorem to bound the distance between the decision $\mathbf{y}_i$ and $\mathbf{y}_j$. As observed earlier, $O, \mathbf{y}_i, \mathbf{d}_i, \mathbf{y}_j$ and I are co-planar. Hyperplanes H_1 and H_{k_i} have the same direction, and contain $\mathbf{y}_i, \mathbf{y}_j$ and $\mathbf{d}_i, I$, respectively. Therefore the lines $(\mathbf{y}_i\mathbf{y}_j)$ and $(\mathbf{d}_i I)$ are parallel. It follows from the Intercept Theorem that

$$\frac{||\mathbf{y}_j - \mathbf{y}_i||_2}{||I - \mathbf{d_i}||_2} = \frac{||\mathbf{y}_i - O||_2}{||\mathbf{d}_i - O||_2}$$

Now, the decision of process p_i is $\mathbf{y}_i = \frac{\mathbf{d}_i}{\sum_{1\leq \ell \leq n} view_i[\ell]} = \frac{\mathbf{d}_i}{k_i}$. Therefore,

$$||\mathbf{y}_j - \mathbf{y}_i||_2 = \frac{||\mathbf{y}_i||_2}{||\mathbf{d}_i||_2}||I - \mathbf{d_i}||_2 = \frac{1}{k_i}\frac{||\mathbf{d}_i||_2}{||\mathbf{d}_i||_2}||I - \mathbf{d_i}||_2 \leq 3\frac{(n-1)}{k_i} \leq 3\frac{(n-1)}{k}.$$

The penultimate inequality comes from Eq. (4), and the last inequality follows from the fact that the sum k_i of the components in the last collect $view_i$ of process p_i is at least k. □

4 Universality of Multidimensional Approximate Agreement

In this section, we show that multidimensional approximate agreement with fixed inputs is complete, in the sense that any problem that is solvable wait-free can be solved by merely solving multidimensional ϵ-agreement for an appropriate setting of ϵ, and inferring the outputs of the problem directly from the solution to multidimensional ϵ-agreement.

To formally define the notion of "problem", it is convenient to adopt the terminology of algebraic topology (see, e.g., [23]). Recall that a *simplicial complex* $\mathcal{K}$ with vertex set V is a collection of non-empty subsets of V containing each singleton $\{v\}$, $v \in V$, and closed by inclusion, i.e., if $\sigma \in \mathcal{K}$ then $\sigma' \in \mathcal{K}$ for every non-empty $\sigma' \subseteq \sigma$. Each set in $\mathcal{K}$ is called a *simplex*. A *task* is then defined as a triple $\Pi = (\mathcal{I}, \mathcal{O}, \Delta)$ where $\mathcal{I}$ and $\mathcal{O}$ are simplicial complexes, and $\Delta : \mathcal{I} \to 2^{\mathcal{O}}$ is the input-output specification. That is, every vertex of $\mathcal{I}$ (resp., of $\mathcal{O}$) is a possible input value (resp., output value), and every $\sigma \in \mathcal{I}$ (resp., $\tau \in \mathcal{O}$) represents a collection of legal input configurations (resp., output configurations) of the system. In other words, if $\sigma = \{x_1, \ldots, x_k\}$ belongs to $\mathcal{I}$, then it is legal that $n \geq k$ processes conjointly start with this set of inputs, i.e., some processes start with input x_1, some others with input x_2, etc. Similarly, if $\tau = \{y_1, \ldots, y_k\}$ belongs to $\mathcal{O}$ then it is legal for a set of $n \geq k$ processes to conjointly output τ, i.e., some processes output y_1, while some others output y_2, etc. Finally, for every $\sigma \in \mathcal{I}$, $\Delta(\sigma)$ is a sub-complex of $\mathcal{O}$ specifying the set of legal outputs for σ, i.e., any set of processes with input configuration σ can collectively output any simplex $\tau \in \Delta(\sigma)$.

For instance, n-dimensional ϵ-agreement with fixed inputs is the task $\Pi = (\mathcal{I}, \mathcal{O}, \Delta)$ with

$$\mathcal{I} = \big\{\{\mathbf{x}_i \mid i \in I\} \mid (I \neq \varnothing) \wedge (I \subseteq \{1, \ldots, n\}\big\}$$

where, for every $i \in \{1, \ldots, n\}$, $\mathbf{x}_i$ is the n-dimensional vector $(0, \ldots, 0, 1, 0, \ldots, 0)$ with the 1 at the i-th coordinate,

$$\mathcal{O} = \big\{\{\mathbf{y}_j \mid j \in J\} \in \mathcal{P}(\mathbb{Q}^n) \mid (\varnothing \neq J \subseteq \{1, \ldots, n\}) \wedge (\forall i, j \in J, ||\mathbf{y}_i - \mathbf{y}_j||_2 \leq \epsilon)\big\}$$

and, for every $\sigma = \{\mathbf{x}_i \mid i \in I\} \in \mathcal{I}$, and every $\tau = \{\mathbf{y}_j \mid j \in J\} \in \mathcal{O}$, we have

$$\tau \in \Delta(\sigma) \iff \mathbf{y}_j \in \mathsf{Hull}(\{\mathbf{x}_i \mid i \in I\}) \text{ for every } j \in J.$$

The following theorem essentially states that an algorithm for multidimensional ϵ-agreement can be used to solve any (solvable) task.

Theorem 16. *Let $n \geq 2$, and let $\Pi = (\mathcal{I}, \mathcal{O}, \Delta)$ be a task solvable by n processes. There exists $\epsilon > 0$ such that, for every input $\sigma = \{x_i \mid i \in I\} \in \mathcal{I}$ for Π, with $\varnothing \neq I \subseteq \{1, \ldots, n\}$, if $\{\mathbf{y}_i \mid i \in I\}$ denotes any solution of multidimensional ϵ-agreement whenever process p_i starts with input $\mathbf{x}_i$ for every $i \in I$, then every process $p_i, i \in I$ can compute locally from $\mathbf{y}_i$ an output y_i for Π such that $\{y_i \mid i \in I\} \in \Delta(\sigma)$.*

Proof. By the colorless wait-free computability theorem [23, 26], since $\Pi = (\mathcal{I}, \mathcal{O}, \Delta)$ is a task solvable by n processes, there exists an integer $T \geq 0$ and a simplicial map[2]

$$f : \mathcal{B}^T(\mathcal{I}) \to \mathcal{O}$$

where $\mathcal{B}^T(\mathcal{I})$ is the simplicial complex obtained by applying T times the barycentric subdivision operator to $\mathcal{I}$ (see Fig. 7). Moreover, this map f agrees with the input-output specification Δ of the task, that is, for every $\sigma \in \mathcal{I}$,

$$f(\mathcal{B}^T(\sigma)) \subseteq \Delta(\sigma),$$

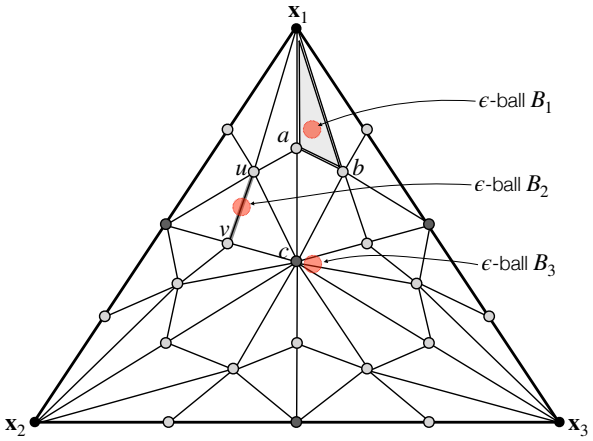

Figure 7: Applying two times the barycentric subdivision operator to $\sigma_\epsilon = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$. Formally, the figure displays the canonical geometric realization of $\mathcal{B}^2(\sigma_\epsilon)$, and the outer triangle represents the frontier of the convex hull of the three points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$.

i.e., every simplex $\sigma' \in \mathcal{B}^T(\sigma)$ is mapped by f to a simplex $f(\sigma')$ of $\Delta(\sigma)$.

Let us denote by $\Pi_\epsilon = (\mathcal{I}_\epsilon, \mathcal{O}_\epsilon, \Delta_\epsilon)$ the multidimensional ϵ-agreement task. Recall that we mean here the fixed inputs version, for which every $i \in \{1, \ldots, n\}$, process p_i can only starts with the point $\mathbf{x}_i$. That is, the input complex of multidimensional ϵ-agreement is simply $\mathcal{I}_\epsilon = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. Now, recall that, by definition of multidimensional ϵ-agreement, the outputs $\mathbf{y}_j, j \in J$, will be at mutual distance at most ϵ, and will stand in the convex hull of the input simplex $\sigma = \{\mathbf{x}_i \mid i \in J\}$.

A first observation is that, by picking ϵ sufficiently small, any ball of radius ϵ in the convex hull of σ can be mapped to a simplex of $\mathcal{B}^T(\sigma)$. For instance, in Fig. 7, the ball B_1 is included in the simplex $\{a, b, \mathbf{x}_1\}$ of $\mathcal{B}^2(\sigma)$. Therefore, each $\mathbf{y}_j$ in B_1 can be mapped to any of the three points a, b, or $\mathbf{x}_1$, e.g., to the closed point. Instead, the ball B_2 intersects a face of $\mathcal{B}^2(\sigma)$, namely the edge $\{u, v\}$. In this case, each $\mathbf{y}_j$ in B_2 can be mapped to any of the two points u or v, e.g., the closest. A ball of radius ϵ may however intersect many different faces, just like the ball B_3 does in Fig. 7. However, for ϵ sufficiently small, this may occur only for balls that are close to a single vertex (c in the case of the ball B_3). Therefore, each $\mathbf{y}_j$ in B_3 can simply be mapped to this vertex. Let us denote by

$$g : \mathcal{O}_\epsilon \to \mathcal{B}^T(\mathcal{I}_\epsilon)$$

this mapping.

A second observation is that there is a canonical one-to-one correspondence between any input simplex $\sigma = \{x_i \mid i \in I\} \in \mathcal{I}$ of the task Π and the input simplex $\sigma_\epsilon = \{\mathbf{x}_i \mid i \in I\} \in \mathcal{I}_\epsilon$ of multidimensional ϵ-agreement. Therefore, the same holds for their barycentric subdivisions. Let us denote by

$$h_\sigma : \mathcal{B}^T(\sigma) \to \mathcal{B}^T(\sigma_\epsilon)$$

this one-to-one map. Note that, for every face σ' of σ, $h_{\sigma'}$ coincides with h_σ restricted to $\mathcal{B}^T(\sigma')$.

We have now all the ingredients for establishing the theorem. Let us first explain how the generic non-negotiating algorithm (Algorithm 1 of Section 1.2) is instantiated by each process. Every process p_j starts by writing its input for task Π (line 1 of the generic algorithm). It then

[2]Recall that a map f from the vertex set of a complex $\mathcal{K}_1$ to the vertex set of a complex $\mathcal{K}_2$ is simplicial if, for every $\sigma \in \mathcal{K}_1$, $f(\sigma) \in \mathcal{K}_2$. Such a map is therefore a map $f : \mathcal{K}_1 \to \mathcal{K}_2$, mapping every simplex of $\mathcal{K}_1$ to a simplex of $\mathcal{K}_2$.

solves multidimensional ϵ-agreement (with fixed input $\mathbf{x}_j$) using the multidimensional approximate agreement algorithm (Algorithm 3). Following algorithm 3, this consists in performing $O(n/\epsilon)$ iterations of the **repeat** loop of the generic algorithm. p_j finally reads all inputs for task Π previously written (line 8 of the generic algorithm.).

By reading the inputs for Π, process p_j thus collects an input simplex $\sigma = \{x_i \mid i \in I\} \in \mathcal{I}$. From σ, p_i infers $\sigma_\epsilon = \{\mathbf{x}_i \mid i \in I\}$ which is its view of the input simplex of the multidimensional approximate agreement. Let $\mathbf{y}$ be the output of p_j in multidimensional ϵ-agreement. Using the map $g : \mathcal{O}_\epsilon \to \mathcal{B}^T(\mathcal{I}_\epsilon)$, process p_j computes

$$\mathbf{z} = g(\mathbf{y}) \in \mathcal{B}^T(\mathcal{I}_\epsilon).$$

Given $\mathbf{z}$, σ, and σ_ϵ, process p_j can then use the one-to-one map $h_\sigma : \mathcal{B}^T(\sigma) \to \mathcal{B}^T(\sigma_\epsilon)$ to compute

$$z = h_\sigma^{-1}(\mathbf{z}) \in \mathcal{B}^T(\sigma).$$

Finally, given $z \in \mathcal{B}^T(\sigma)$, process p_j just outputs $y = f(z)$ where $f : \mathcal{B}^T(\mathcal{I}) \to \mathcal{O}$ is the aforementioned simplicial map whose existence is guaranteed by the wait-free computability theorem.

To show correctness, let $I \subseteq \{1, \ldots, n\}$ be the set of (correct) participating processes, with input simplex $\{x_i \mid i \in I\}$. These processes solve multidimensional ϵ-agreement with input $\sigma_\epsilon = \{\mathbf{x}_i \mid i \in I\}$, and output $\{\mathbf{y}_j \mid j \in J\} \in \Delta_\epsilon(\sigma_\epsilon)$. Thanks to the mapping g, these output values are mapped to a simplex $\tau \in \mathcal{B}^T(\sigma_\epsilon)$, which is, thanks to h_σ, in one-to-one correspondence with a simplex $\tau' \in \mathcal{B}^T(\sigma)$. Since f is simplicial, the latter simplex is mapped to a simplex $\tau'' \in \mathcal{O}$. In fact, since f solves the task Π, and since $\tau' \in \mathcal{B}^T(\sigma)$, we necessarily have $\tau'' \in \Delta(\sigma)$ as f agrees with Δ. This completes the proof. □

The following is a direct consequence of Theorem 16, merely because multidimensional ϵ-agreement with process p_i starting with input $\mathbf{x}_i$ for every $i \in \{1, \ldots, n\}$ can be solved using a non-negotiating algorithm.

Corollary 17. *Let $n \geq 2$, and let $\Pi = (\mathcal{I}, \mathcal{O}, \Delta)$ be a task solvable by n processes with an unrestricted algorithm. There exists a non-negotiating algorithm solving Π for n processes.*

5 Conclusion

We have introduced a very restricted form distributed wait-free shared memory computing, and shown that nevertheless, it is universal, in the sense that it is capable of solving any colorless task that is wait-free solvable under no restrictions on the algorithm. The result is achieved by proving a general result about multidimensional approximate agreement: a black box that solves this task for arbitrary $\epsilon > 0$, can be used to solve any (wait-free solvable) colorless task.

Our results open several interesting avenues for future research. They uncover the remarkable power of asynchronous read/write shared memory, that allows processes to communicate to each other information, through the timing of their read/write operations, which is not under their control. Although we proved a lower bound for two process uniform algorithms, in general we don't know if the exponential slowdown of our n-processes multidimensional approximate agreement algorithm is unavoidable.

It would also be interesting to compare our results to dynamic graph message passing [16] or shared-memory iterated models [31], where processes operate in rounds, and hence the round number is a counter available for free. For instance, it has been shown [9] that the wait-free dynamic graph model is strong enough to implement any two process task, without an exponential slowdown, even using only *beeps*. Interestingly, in this and other beeping models

processes decide when to send a beep, as a function of their history. Our non-negotiating model is more restrictive in this sense, and indeed beeps would be useless; the moment a process writes is not under its own control, since the model is fully asynchronous.

The class of tasks we have considered are *colorless* [23], which includes consensus, approximate agreement, set agreement, and many others. It remains an open question if our non-negotiating model can be used to solve colored tasks, most notably, renaming [6].

References

[1] Yehuda Afek, Noga Alon, Omer Barad, Eran Hornstein, Naama Barkai, and Ziv Bar-Joseph. A biological solution to a fundamental distributed computing problem. *science*, 331(6014):183–185, 2011.

[2] Bertie Ancona, Ayesha Bajwa, Nancy A. Lynch, and Frederik Mallmann-Trenn. How to color a french flag - biologically inspired algorithms for scale-invariant patterning. In *14th Latin American Symposium on Theoretical Informatics (LATIN)*, volume 12118 of *LNCS*, pages 413–424. Springer, 2020.

[3] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Comput.*, 18(4):235–253, 2006.

[4] Hagit Attiya and Faith Ellen. The step complexity of multidimensional approximate agreement. In *26th International Conference on Principles of Distributed Systems, OPODIS*, volume 253 of *LIPIcs*, pages 6:1–6:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.OPODIS.2022.6.

[5] Hagit Attiya and Sergio Rajsbaum. The combinatorial structure of wait-free solvable tasks. *SIAM J. Comput.*, 31(4):1286–1313, 2002.

[6] Hagit Attiya and Jennifer Welch. *Distributed computing: fundamentals, simulations, and advanced topics*, volume 19. John Wiley & Sons, 2004.

[7] Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *24th International Symposium on Distributed Computing (DISC)*, volume 6343 of *LNCS*, pages 148–162. Springer, 2010.

[8] Peter Davies. Optimal message-passing with noisy beeps. In *42th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 300–309, 2023.

[9] Carole Delporte-Gallet, Hugues Fauconnier, and Sergio Rajsbaum. Communication complexity of wait-free computability in dynamic networks. In *27th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 12156 of *LNCS*, pages 291–309. Springer, 2020.

[10] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. In *3rd IEEE Symposium on Reliability in Distributed Software and Database Systems (SRDS)*, pages 145–154, 1983.

[11] El-Mahdi El-Mhamdi, Rachid Guerraoui, Arsany Guirguis, Lê-Nguyên Hoang, and Sébastien Rouault. Genuinely distributed byzantine machine learning. *Distributed Computing*, 35(4):305–331, 2022.

[12] Yuval Emek and Roger Wattenhofer. Stone age distributed computing. In *32nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 137–146, 2013.

[13] Ofer Feinerman, Bernhard Haeupler, and Amos Korman. Breathe before speaking: efficient information dissemination despite noisy, limited and anonymous communication. *Distributed Comput.*, 30(5):339–355, 2017.

[14] Ofer Feinerman and Amos Korman. The ANTS problem. *Distributed Comput.*, 30(3):149–168, 2017.

[15] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[16] Matthias Függer, Thomas Nowak, and Manfred Schwarz. Tight bounds for asymptotic and approximate consensus. *J. ACM*, 68(6):46:1–46:35, 2021. doi:10.1145/3485242.

[17] Eli Gafni and Elias Koutsoupias. Three-processor tasks are undecidable. *SIAM J. Comput.*, 28(3):970–983, 1999.

[18] Aviram Gelblum, Ehud Fonio, Yoav Rodeh, Amos Korman, and Ofer Feinerman. Ant collective cognition allows for efficient navigation through disordered environments. *eLife*, 9:e55195, may 2020. doi:10.7554/eLife.55195.

[19] Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Multidimensional approximate agreement with asynchronous fallback. In *35th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 141–151, 2023.

[20] George Giakkoupis and Isabella Ziccardi. Distributed self-stabilizing MIS with few states and weak communication. In *42nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 310–320, 2023.

[21] Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. *Distributed Computing*, 28(1):31–53, 2015.

[22] Maurice Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1):124–149, 1991. doi:10.1145/114005.102808.

[23] Maurice Herlihy, Dmitry N. Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, 2013.

[24] Maurice Herlihy and Sergio Rajsbaum. The decidability of distributed decision tasks. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, pages 589–598. ACM, 1997. doi:10.1145/258533.258652.

[25] Maurice Herlihy and Sergio Rajsbaum. The topology of shared-memory adversaries. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, page 105–113. ACM, 2010. doi:10.1145/1835698.1835724.

[26] Maurice Herlihy, Sergio Rajsbaum, Michel Raynal, and Julien Stainer. From wait-free to arbitrary concurrent solo executions in colorless distributed computing. *Theor. Comput. Sci.*, 683:1–21, 2017.

[27] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999.

[28] Gunnar Hoest and Nir Shavit. Toward a topological characterization of asynchronous complexity. *SIAM J. Comput.*, 36(2):457–497, 2006.

[29] Hammurabi Mendes, Maurice Herlihy, Nitin H. Vaidya, and Vijay K. Garg. Multidimensional agreement in byzantine systems. *Distributed Comput.*, 28(6):423–441, 2015.

[30] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.

[31] Sergio Rajsbaum. Iterated shared memory models. In Alejandro López-Ortiz, editor, *9th Latin American Symposium on Theoretical Informatics (LATIN)*, volume 6034 of *Lecture Notes in Computer Science*, pages 407–416. Springer, 2010. doi:10.1007/978-3-642-12200-2_36.

[32] Michel Raynal. *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*. Springer, 2018. doi:10.1007/978-3-319-94141-7.

[33] Gadi Taubenfeld. Anonymous shared memory. *J. ACM*, 69(4), 2022.