Chapter 20

Approximate Agreement

Imagine a cooling room where a group of nodes equipped with thermometers are measuring the room's temperature. Each node is located in a different part of the room, and the thermometers are not perfectly accurate. Can the nodes agree on the room's temperature, even if some of the nodes are byzantine?

While byzantine agreement, as discussed in Chapter 17, offers an immediate solution, it comes with certain limitations. For instance, the standard validity definition (all-same validity, Definition 17.5) allows the output value a corrupted value unless correct nodes measure precisely the same temperature. On the other hand, Median Validity (Definition 17.6) would ensure that the agreed-upon temperature is close to the median of the correct measurements. This is, in fact, an excellent guarantee. However, for this chapter, a weaker definition will suffice.

Definition 20.1 (Correct-Range Validity)**.** *Correct nodes' outputs fall within the range of correct nodes' inputs.*

Moreover, deterministic synchronous byzantine agreement algorithms, with or without strong validity guarantees, are inherently slow. On top of that, the synchronous model's assumptions may be a bit too strong for a real-life network, and asynchronous algorithms are randomized and complicated.

In this chapter, we will learn how to overcome these limitations by relaxing the agreement property of byzantine agreement. That is, in the real world, does it really matter if a node believes that the agreed-upon temperature is $25.1276°C$ and another node believes it is $25.1277°C$? We will allow the nodes to agree on a temperature up to a very small error $\varepsilon > 0$.

Definition 20.2 (ε-Agreement)**.** *If two correct nodes output x and y, then $|x - y| \leq \varepsilon$.*

This defines an exciting variant of byzantine agreement, known as *Approximate Agreement.*

Definition 20.3 (Approximate Agreement)**.** *There are n nodes, of which f might be byzantine. Every node holds an input value in $\mathbb{R}$. For any predefined $\varepsilon > 0$, every correct node must output a value in $\mathbb{R}$ such that correct-range validity and ε-agreement hold.*

20.1 How many corruptions can we tolerate?

Theorem 20.4. *Even in the synchronous model, there is no algorithm achieving approximate agreement when $n \leq 3f$.*

Proof Sketch. Assume that there is an algorithm A that achieves approximate agreement when $n \leq 3f$. We partition the n nodes into three (non-empty) sets of size at most f: V_0, V_1, and V_b. Nodes in set V_0 are correct and have input 0, and, similarly, nodes in set V_1 are correct and have input 1. The nodes in V_b are corrupted, and, similarly to Theorem 17.13, they support the input value of each correct node. This way, because of correct-range validity, nodes in V_0 output 0, and nodes in V_1 output 1, which breaks ε-agreement for any $\varepsilon < 1$. □

Remarks:

- Can we show that this bound is tight in the synchronous model?

20.2 Synchronous Algorithm

Algorithm 20.5 Synchronous Approximate Agreement

```
1: Code for node $v$ with input $x$.
2: $I = \lceil \log_2(\texttt{max\_range}/\varepsilon) \rceil$.
3: $x_0 = x$.
4: for $i$ in $1...I$ do
5:     Send $x_{i-1}$ to all nodes.
6:     Add every received value to $R_i$.
7:     $T_i$ = the multiset obtained by removing the lowest $f$ values in $R_i$ and the
       highest $f$ values in $R_i$.
8:     $x_i = (\min T_i + \max T_i)/2$.
9: end for
10: Output $x_I$.
```

Remarks:

- We will assume that the input space is bounded, i.e. the honest nodes' inputs are stored as `double` variables. The variable `max_range` represents the size of the input space. Approximate agreement can also be solved without this assumption: through a mechanism that allows correct nodes to estimate the initial range.
- R_i and T_i are *multisets*, i.e., sets with repeated values.
- In every iteration i, the correct nodes' goal is to compute values x_i that get closer and are within the range of correct values x_{i-1}. This way, after sufficiently many iterations, ε-agreement is achieved.

Lemma 20.6. *Assume $n > 3f$, and let X be a multiset of $n - f$ values (intuitively, representing the correct values). Let R denote a multiset containing $n - f + k$ values, with $0 \leq k \leq f$, such that $|X \cap R| \geq n - 2f + k$.*

Then, if a multiset T is obtained by removing the lowest f and the highest f values in R, it satisfies $T \neq \emptyset$ and $T \subseteq [\min X, \max X]$.

Proof. We first show that T is non-empty: $|T| = |R| - 2f \geq n - 3f > 0$.

We now focus on showing that T is included in $[\min X, \max X]$. R contains at most values f in addition to those in $X \cap R$. Only these f values may be lower than $\min X$ or higher than $\max X$. Since T is obtained by removing the lowest f and the highest f values from R, $\min X \leq \min T$ and $\max T \leq \max X$. □

Lemma 20.7. *Let T and T' denote two multisets such that $T \cap T' \neq \emptyset$, and let $x = (\min T + \max T)/2$ and $y = (\min T' + \max T')/2$.*

Then, $|x - y| \leq (\max(T \cup T') - \min(T \cup T'))/2$.

Proof. We assume without loss of generality that $y \geq x$. Since $T \cap T' \neq \emptyset$, $\min T' \leq \max T$, which allows us to obtain the following:

$$\begin{aligned} y - x &= (\max T' + \min T')/2 - (\max T + \min T)/2 \\ &\leq (\max T' - \min T)/2 \leq (\max(T \cup T') - \min(T \cup T'))/2. \end{aligned}$$

□

Lemma 20.8. *Assume $n > 3f$. Let R and R' denote two multisets of at most n values such that $|R \cap R'| \geq 2f + 1$. Compute T and T' by removing the lowest f and the highest f values of R and R' respectively. Then, $T \cap T' \neq \emptyset$.*

Proof. Let $R_\cap = R \cap R'$. Since $R_\cap \subseteq R$, the multiset $T_\cap$ obtained by removing the lowest f and the highest f values of $R_\cap$ satisfies $T_\cap \subseteq T$. Similarly, $T_\cap$ is also included in T', and therefore $T_\cap \subseteq T \cap T'$.

It remains to show that $T_\cap$ is non-empty: $|T_\cap| = |R_\cap| - 2f \geq 1$. □

Theorem 20.9. *Algorithm 20.5 achieves approximate agreement secure against $f < n/3$ byzantine corruptions.*

Proof. Let X_0 denote the multiset containing the correct nodes' input values, and let X_i denote the multiset containing the values x_i obtained by the correct nodes in iteration i. We use induction on $0 \leq i \leq I$ to show that Algorithm 20.5 provides following properties: every correct node obtains a value $x_i \in [\min X_{i-1}, \max X_{i-1}]$, and $\max X_i - \min X_i \leq (\max X_0 - \min X_0)/2^i$.

The base case is trivial: nodes set x_0 to their inputs. For the induction step, assume that the properties hold for $i - 1$, and we show that they also hold for i:

- Every correct node holds a value $x_i \in [\min X_{i-1}, \max X_{i-1}]$: Since every correct node holds a value x_{i-1} at the beginning of iteration i, every node obtains a multiset R_i containing $n - f$ values in X_{i-1} (from the correct nodes, as the network is synchronous), and at most f byzantine values.

  Applying Lemma 20.6, we obtain that every correct node obtains a multiset $T_i \subseteq [\min X_{i-1}, \max X_{i-1}]$, and a value $x_i \in [\min X_{i-1}, \max X_{i-1}]$.

- $\max X_i - \min X_i \leq (\max X_0 - \min X_0)/2^i$: Let x_i and y_i denote the values obtained by two correct nodes v and u in iteration i. We use Lemma 20.7 to show that $|x_i - y_i| \leq (\max X_{i-1} - \min X_{i-1})/2$.

  Nodes v and u have obtained multisets R_i that intersect in $2f + 1$ values: they both contain the $n - f \geq 2f + 1$ correct values in X_{i-1}. Lemma 20.8

then ensures that the multisets T_i and T_i' obtained by v and u respectively intersect as well. We may then apply Lemma 20.7, which ensures that $|x_i - y_i| \leq (\max(T_i \cup T_i') - \min(T_i \cup T_i'))/2$. In addition, according to Lemma 20.6, $T_i, T_i' \subseteq [\min X_{i-1}, \max X_{i-1}]$. We can conclude that:

$$|x_i - y_i| \leq (\max X_{i-1} - \max X_{i-1})/2 \leq (\max X_0 - \max X_0)/2^i.$$

We have obtained that, in every iteration, nodes hold values satisfying correct-range validity. In addition, after $\lceil \log_2((\max X_0 - \min X_0)/\varepsilon) \rceil \leq I$ iterations, ε-agreement was already achieved, and the following iterations maintained it. Therefore, Algorithm 20.5 achieves approximate agreement. □

Remarks:

- What about asynchronous communication?

20.3 Asynchronous Algorithm

Algorithm 20.10 Asynchronous Approximate Agreement: Naive Attempt

```
1: Code for node $v$ with input $x$.
2: $I = \lceil \log_2(\texttt{max\_range}/\varepsilon) \rceil$.
3: $x_0 = x$.
4: for $i$ in 1...$I$ do
5:    Send $\texttt{msg}_i(x_{i-1})$ to all nodes.
6:    upon receiving $\texttt{msg}_i(y_{i-1})$ from $u$:
7:       If this is the first message $\texttt{msg}_i$ from $u$, add $y_{i-1}$ to $R_i$.
8:       When $R_i$ contains values from $n - f$ nodes:
9:           $T_i$ = the multiset obtained by removing the lowest $f$ values in $R_i$
             and the highest $f$ values in $R_i$.
10:          $x_i = (\min T_i + \max T_i)/2$.
11:          Start the next iteration.
12:   end upon
13: end for
14: Output $x_I$.
```

Remarks:

- Does Algorithm 20.10 achieve approximate agreement when $f < n/3$? No.

  Counterexample: Assume $n = 4$ and $f = 1$. Nodes v_0, v_1, v_2 are correct and have inputs $0, 1, 1$ respectively. The fourth node v_b is byzantine. In every iteration, the byzantine node v_b sends -1 to v_0, and nothing to v_1 and v_2. We delay any message v_2 sends to v_0. Hence, in the first iteration, v_0 obtains $R_1 = \{-1, 0, 1\}$ and therefore computes $x_1 = 0$. On the other hand, both v_1 and v_2 obtain $R_1' = \{0, 1, 1\}$, and therefore compute $x_1 = 1$. Each correct node maintains its input value, and correct values never get ε-close for any $\varepsilon < 1$.

- What about $f < n/4$? Also no.

  Counterexample: Assume $n = 5$ and $f = 1$. Nodes v_0, v_1, v_2, v_3 are correct and have inputs $0, 0, 1, 1$ respectively. The fifth node v_b is byzantine. In every iteration, nodes v_b sends -1 to v_0 and v_1, and 2 to v_2 and v_3. The messages v_0 sends to v_2 and v_3 are delayed, and, similarly, the messages v_3 sends to v_0 and v_1 are delayed. Hence, in the first iteration, both v_0 and v_1 obtain $R_1 = \{-1, 0, 0, 1\}$, while v_2 and v_3 obtain $R'_1 = \{0, 1, 1, 2\}$. Hence, just like in the previous counterexample, correct nodes maintain their input values.

- Does Algorithm 20.10 achieve approximate agreement when $f < n/5$? Yes. ε-agreement holds now: the multisets R_i pair-wise intersect in $2f+1$ values, which enables us to apply Lemma 20.8 and Lemma 20.7.

- To break ε-agreement when $f < n/4$, the byzantine nodes send inconsistent values. Is there any way we could prevent this?

- Reliable Broadcast (Algorithm 18.11) comes with a valuable property. Regardless of whether the sender is correct or not, if a node v accepts a value, all other nodes accept the same value eventually. Algorithm 18.11, however, allows nodes to accept multiple values.

20.3.1 Single-Value Reliable Broadcast

Algorithm 20.11 Single-Value Reliable Broadcast: Iteration i, Sender v_S

```
Code for sender $v_S$ with input $x_S$:
Send msg$_{<i,v_S>}(x_S)$ to everyone.

Code for node $v$:
// Ignore any messages tagged with different values $i, v_S$.
upon receiving msg$_{<i,v_S>}(x)$ from $v_S$:
  If no echo$_{<i,v_S>}$ message was sent in this instance:
      Send echo$_{<i,v_S>}(x)$ to everyone.
end upon

upon receiving echo$_{<i,v_S>}(x)$ from $n - f$ distinct nodes or
               ready$_{<i,v_S>}(x)$ from $f + 1$ distinct nodes:
  Send ready$_{<i,v_S>}(x)$ to everyone.
end upon

upon receiving ready$_{<i,v_S>}(x)$ from $2f + 1$ distinct nodes:
  Accept msg$_{<i,v_S>}(x)$.
end upon
```

Theorem 20.12. *Algorithm 20.11 achieves the following properties, even when $f < n/3$ of the nodes involved are byzantine:*

- *If the sender v_S is correct, every correct node accepts* $\texttt{msg}_{<i,v_S>}(x_S)$.

- *If a correct node accepts* $\texttt{msg}_{<i,v_S>}(x)$*, then every correct node accepts* $\texttt{msg}_{<i,v_S>}(x)$ *eventually, and no correct node accepts* $\texttt{msg}_{<i,v_S>}(y)$ *with* $y \neq x$*.*

Remarks:

- The tag $< i, v_S >$ represents the instance's identifier. This enables nodes to distinguish which messages belong to which algorithm, which is very helpful when composing algorithms. Most often, for simplicity of presentation, the identifiers are implicit.
- Messages with differents tag $< i', v'_S >$ are ignored in an instance i with sender v_S. You can think of them as being stored in some queue until instance i with sender v_S is running.

Lemma 20.13. *Assume that the sender* v_S *is correct and has input* x_S*. Then, every correct node accepts* $\texttt{msg}_{<i,v_S>}(x_S)$*.*

Proof. First, since v_S is correct, no correct node sends $\texttt{echo}_{<i,v_S>}(y)$ for any value $y \neq x_S$, and hence no correct node sends $\texttt{ready}_{<i,v_S>}(y)$ for any $y \neq x_S$. Therefore, no correct node accepts $y \neq x_S$.

Every node eventually receives $\texttt{msg}_{<i,v_S>}(x_S)$ from the correct sender, and therefore every correct node eventually sends $\texttt{echo}_{<i,v_S>}(x_S)$. It follows that every correct node eventually receives $n-f$ messages $\texttt{echo}_{<i,v_S>}(x_S)$ and hence sends $\texttt{ready}_{<i,v_S>}(x_S)$. Finally, every correct node eventually receives $n-f$ messages $\texttt{ready}_{<i,v_S>}(x_S)$ and accepts $\texttt{msg}_{<i,v_S>}(x_S)$. □

Lemma 20.14. *If a correct node* v *sends* $\texttt{ready}_{<i,v_S>}(x)$*, then no correct node sends* $\texttt{ready}_{<i,v_S>}(y)$ *for* $y \neq x$*.*

Proof. Without loss of generality, assume that v and u are the first correct nodes that send $\texttt{ready}_{<i,v_S>}(x)$ and $\texttt{ready}_{<i,v_S>}(y)$ respectively. Then, v has received $\texttt{echo}_{<i,v_S>}(x)$ from $n-f$ nodes, while u has received $\texttt{echo}_{<i,v_S>}(y)$ from $n-f$ nodes. Then, there are $(n-f)+(n-f)-n > f$ nodes, hence at least one correct node, that sent multiple echo messages for different values, which contradicts the algorithm. □

Lemma 20.15. *If correct nodes* v *and* u *accept* $\texttt{msg}_{<i,v_S>}(x)$ *and* $\texttt{msg}_{<i,v_S>}(y)$ *respectively, then* $x = y$*.*

Proof. Since v has accepted $\texttt{msg}_{<i,v_S>}(x)$, at least one correct node has sent $\texttt{ready}_{<i,v_S>}(x)$. Then, no correct node has sent $\texttt{ready}_{<i,v_S>}(y)$ for $y \neq x$, according to Lemma 20.14. Therefore, if u accepts $\texttt{msg}_{<i,v_S>}(y)$, $x = y$. □

Lemma 20.16. *If a correct node* v *accepts* $\texttt{msg}_{<i,v_S>}(x)$*, then every correct node accepts* $\texttt{msg}_{<i,v_S>}(x)$ *eventually.*

Proof. By Lemma 20.15, no correct node accepts $\texttt{msg}_{<i,v_S>}(y)$ with $y \neq x$.

Node v has received $2f+1$ messages $\texttt{ready}_{<i,v_S>}(x)$, hence at least $f+1$ messages $\texttt{ready}_{<i,v_S>}(x)$ coming from correct nodes. All correct nodes eventually receive these $f+1$ messages $\texttt{ready}_{<i,v_S>}(x)$. As Lemma 20.14 guarantees that no correct node sends $\texttt{ready}_{<i,v_S>}(y)$ for $y \neq x$, it follows that every correct node sends $\texttt{ready}_{<i,v_S>}(x)$ eventually. These messages are delivered eventually, and therefore all correct nodes accept the same $\texttt{msg}_{<i,v_S>}(x)$. □

Algorithm 20.17 Aynchronous Approximate Agreement: Second Attempt

```
1: Code for node $v$ with input $x$.
2: $I = \lceil \log_2(\texttt{max\_range}/\varepsilon) \rceil$.
3: $x_0 = x$.
4: for $i$ in 1...$I$ do
5:     Send $x_{i-1}$ to all nodes via Algorithm 20.11 (in the instance for iteration
       $i$, with sender $v$).
6:     upon accepting $\texttt{msg}_{<i,u>}(y_{i-1})$ from $u$ via Algorithm 20.11 (that is, in
       the instance of iteration $i$ with sender $u$):
7:         Add $y_{i-1}$ to $R_i$.
8:         When $R_i$ contains values from $n - f$ nodes:
9:             $T_i$ = the multiset obtained by removing the lowest $f$ values in $R_i$
               and the highest $f$ values in $R_i$.
10:            $x_i = (\min T_i + \max T_i)/2$.
11:            Start the next iteration.
12:    end upon
13: end for
14: Output $x_I$.
```

Remarks:

- Does Algorithm 20.10 achieve approximate agreement secure against $f < n/4$ corruptions? Yes.

  Byzantine nodes cannot send inconsistent values anymore. Even when $f < n/4$, nodes obtain multisets R_i that pair-wise intersect in at least $2f + 1$ values. This allows us to prove that ε-agreement holds with the help of Lemma 20.8 and Lemma 20.7.

- Does Algorithm 20.10 achieve approximate agreement secure against $f < n/3$ corruptions? Unfortunately not.

  Counterexample: Assume $n = 4$ and $f = 1$. Nodes v_0, v_1, v_2 are correct and have inputs $0, 1, 1$ respectively. The fourth node v_b is byzantine. In every iteration, the byzantine node v_b sends -1 via Algorithm 20.11. Node v_0 is the first correct node to receive this value. For nodes v_1 and v_2, this value is delayed. Hence, although v_b sends its value via Algorithm 20.11, v_1 and v_2 will not receive it *fast enough.*

  We similarly delay any message v_2 sends to v_0, even though it is sent via Algorithm 20.11. Hence, in the first iteration, v_0 obtains $R_1 = \{-1, 0, 1\}$ and therefore computes $x_1 = 0$. On the other hand, both v_1 and v_2 obtain $R'_1 = \{0, 1, 1\}$, and therefore compute their new values as $x_1 = 1$. In each of the following iterations, the correct nodes will compute their new values identically.

- The main issue behind our attempts so far is that, if $f < n/3$, the multisets T_i do not necessarily pair-wise intersect. This may prevent the correct values from converging. Having more values in common in the multisets R_i would help us, as suggested by Lemma 20.8.

- If only v_0 could tell v_1 and v_2 to wait a bit longer for v_b's value... The value sent by v_b cannot be delayed forever for the other nodes.
- We just need to convince nodes to wait long enough. But *what does long enough mean?* The so-called *Witness Technique* can help us.

20.3.2 The Witness Technique

Algorithm 20.18 The Witness Technique: Iteration i

```
1: Code for node $v$ with input $x$.
2: Let $R = \emptyset$, $S = \emptyset$, $W = \emptyset$.
3: Send $x$ to all the nodes via Algorithm 20.11 (in the instance for iteration $i$,
   with sender $v$).
4: upon accepting $\texttt{msg}_{i,u}(y)$ from $u$ via Algorithm 20.11 (in the instance for
   iteration $i$ with sender $u$):
5:    Add $y$ to $R$ and $u$ to $S$.
6:    The first time when $|S| \geq n - f$ holds:
7:        Send $\texttt{wait}_i(S)$ to all the nodes.
8: end upon
9: upon receiving $\texttt{wait}_i(S')$ from $u$ such that $|S'| \geq n - f$:
10:   When $S' \subseteq S$, add $u$ to $W$.
11:   The first time when $|W| \geq n - f$:
12:       Output $R$.
13: end upon
```

Remarks:

- Once a node accepts values from $n - f$ distinct nodes via Algorithm 20.11, it will report the set of senders S: *"I got values from these nodes, therefore you can wait for them as well".*
- When a node v receives such a set S' from u, v checks if it has obtained values from the nodes in S' as well. If this is the case, v marks u as a witness. Otherwise, v will keep waiting, and it will receive more values via Algorithm 20.11 (so eventually v can mark u as a witness) or more sets S'. Once v marks $n - f$ nodes as witnesses, v can stop waiting for values and output R.
- Waiting for $n - f$ witnesses will ensure that v and any node u have at least one correct witness in common. This correct witness has convinced v and u to wait for the same $n - f$ values. Hence, their multisets T_i will intersect, which leads to convergence.

Lemma 20.19. *Assume that a correct node v has output R in Algorithm 20.18. Then, R contains at least $n-f+k$ values, with $0 \leq k \leq f$, out of which $n-2f+k$ are sent by correct nodes.*

Proof. Node v adds to R any value received via Algorithm 20.11. As outputting R requires that $|W| \geq n - f$, we obtain that $|S| \geq n - f$, i.e. R contains the values that v received from $n - f + k$ nodes (with $0 \leq k \leq f$). Out of these, at most f will be sent by byzantine nodes, while the remaining are sent by correct nodes (which Algorithm 20.11 ensures to be received correctly). □

Lemma 20.20. *Let v and u denote two correct nodes, and assume they output R and R' respectively. Then, $|R \cap R'| \geq n - f$.*

Proof. Since v and u have obtained outputs, the termination condition of Algorithm 20.18 ensures that they have obtained sets W and resp. W' such that $|W|, |W'| \geq n - f$. Then, v and u have $f + 1$ common witnesses: $|W \cap W'| \geq (n - f) + (n - f) - n = n - 2f > f$. At least one of these common witnesses is a correct node w.

Node w has sent the same set S_w to both v and u. Both v and u have consistently received all the values sent by the $n - f$ nodes in S_w via Algorithm 20.11 and added them to R and R' respectively. Therefore, $|R \cap R'| \geq n - f$. □

Lemma 20.21. *Every correct node v eventually outputs R.*

Proof. It is sufficient to show that v eventually obtains $|W| \geq n - f$, i.e., marks $n - f$ nodes as witnesses. In the following, we show that node v marks all correct nodes as witnesses (unless $|W| \geq n - f$ already holds). Every node receives $n - f$ values via Algorithm 20.11 eventually (as there are $n - f$ correct nodes). Hence, every correct node sends its set S eventually. Node v eventually receives the set S' from a correct node u, and v eventually obtains outputs in the instances of Algorithm 20.11 having nodes in S' as senders. Therefore, v eventually adds u to W. Since this applies to every correct node, it eventually holds that $|W| \geq n - f$. □

20.3.3 Optimal-Resilience Asynchronous Algorithm

Algorithm 20.22 Aynchronous Approximate Agreement

```
1: Code for node $v$ with input $x$.
2: $I = \lceil \log_2(\texttt{max\_range}/\varepsilon) \rceil$.
3: $x_0 = x$.
4: for $i$ in 1...$I$ do
5:     Send $x_{i-1}$ to all nodes via Algorithm 20.18 (in the unique instance corre-
       sponding to iteration $i$).
6:     upon obtaining output $R_i$:
7:         $T_i$ = the multiset obtained by removing the lowest $f$ values in $R_i$ and
           the highest $f$ values in $R_i$.
8:         $x_i = (\min T_i + \max T_i)/2$.
9:         Start the next iteration.
10:    end upon
11: end for
12: Output $x_I$.
```

Theorem 20.23. *Algorithm 20.22 achieves asynchronous approximate agreement secure against $f < n/3$ byzantine corruptions.*

Proof. The proof is similar to that of Theorem 20.9. We use X_0 to denote the multiset containing the correct nodes' input values, and X_i to denote the multiset containing the values x_i obtained by the correct nodes in iteration i. Using induction on $0 \leq i \leq I$, one can show that Algorithm 20.22 provides the following properties: every correct node obtains a value $x_i \in [\min X_{i-1}, \max X_{i-1}]$, and $\max X_i - \min X_i \leq (\max X_0 - \min X_0)/2^i$. This will then imply that Algorithm 20.22 achieves approximate agreement.

The base case $i = 0$ is trivial: nodes initialize x_0 to their inputs. For the induction step, assume that the properties hold for $i - 1$, and we show that they also hold for i:

- Every correct node holds a value $x_i \in [\min X_{i-1}, \max X_{i-1}]$: Lemma 20.21 ensures that every correct node obtains a multiset R_i via Algorithm 20.18. Lemma 20.19 enables us to apply Lemma 20.6, and obtain that every correct node obtains a multiset $T_i \subseteq [\min X_{i-1}, \max X_{i-1}]$, and a value $x_i \in [\min X_{i-1}, \max X_{i-1}]$.

- $\max X_i - \min X_i \leq (\max X_0 - \min X_0)/2^i$: Let x_i and y_i denote the values obtained by two correct nodes v and u in iteration i. We use Lemma 20.7 to show that $|x_i - y_i| \leq (\max X_{i-1} - \min X_{i-1})/2$.

  Nodes v and u have obtained multisets R_i and resp. R'_i that intersect in $n - f$ values, according to Lemma 20.20. Then, we may apply Lemma 20.8 and obtain that the multisets T_i and T'_i have a non-empty intersection. Then, Lemma 20.7 ensures that $|x_i - y_i| \leq (\max(T_i \cup T'_i) - \min(T_i \cup T'_i))/2$. In addition, according to Lemma 20.6, $T_i, T'_i \subseteq [\min X_{i-1}, \max X_{i-1}]$. This enables us to conclude that:

  $$|x_i - y_i| \leq (\max X_{i-1} - \min X_{i-1})/2 \leq (\max X_0 - \min X_0)/2^i.$$

□

Chapter Notes

While approximate agreement provides weaker guarantees in comparison to byzantine agreement, it comes with many advantages: fast synchronous algorithms, and simple deterministic solutions in the asynchronous model. In fact, many real-world scenarios that involve floating-point values are inherently prone to small errors. Notable examples include clock synchronization [HSSD84, LMS85, WL88], and robot coordination techniques, such as line-gathering algorithms [BPBT10].

Approximate agreement was introduced in 1986 by Dolev, Lynch, Pinter, Stark and Weihl [DLP$^+$86]. In this paper, they show how Approximate Agreement can be achieved when $f < n/5$ in an asynchronous network, and up to the optimal threshold $f < n/3$ when the network is synchronous. Later, Abraham, Amit, and Dolev [AAD05] have shown that the condition $f < n/3$ is sufficient in the asynchronous model as well, and have proposed the Witness Technique.

The literature has considered various extensions of approximate agreement, which address a wider range of scenarios Some of these are *multidimensional approximate agreement*, introduced by Mendes, Herlihy, Vaidya, and Garg [MH13, VG13]. In this variant, each node holds a vector in $\mathbb{R}^D$ as input, and the correct parties try to converge to ε-close (in terms of Euclidean distance) outputs in $\mathbb{R}^D$ that lie in the convex hull of their inputs.

Considering higher dimensions also turns out to be relevant in several practical applications, including scenarios where robots need to converge to close locations in a 2 or 3-dimensional space [PBRT11], in distributed voting where the preferences are described by assigning weights, or in optimization problems, and maybe most prominently in machine learning [EMGG$^+$20, SV16]: in federated machine learning, n parties (e.g., companies, hospitals) want to (or, must) keep their training data private, but they agree to improve their model based on the data of other parties. So each party runs its own machine learning model, and learns with its own data. From time to time, the parties exchange their learning parameters (in particular gradients, which are vectors). The parties try to approximately agree on a gradient, while being resilient to Byzantine faults.

Bibliography

[AAD05] 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'04, pages 229–239, Berlin, Heidelberg, 2005. Springer-Verlag.

[BPBT10] Zohir Bouzid, Maria Gradinariu Potop-Butucaru, and SÃľbastien Tixeuil. Optimal byzantine-resilient convergence in unidimensional robot networks. *Theoretical Computer Science*, 411(34):3154–3168, 2010.

[DLP$^+$86] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499âĂŞ516, May 1986.

[EMGG$^+$20] El-Mahdi El-Mhamdi, Rachid Guerraoui, Arsany Guirguis, Lê Nguyên Hoang, and Sébastien Rouault. Genuinely distributed byzantine machine learning. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20, page 355âĂŞ364, New York, NY, USA, 2020. Association for Computing Machinery.

[HSSD84] Joseph Y. Halpern, Barbara Simons, Ray Strong, and Danny Dolev. Fault-tolerant clock synchronization. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, page 89âĂŞ102, New York, NY, USA, 1984. Association for Computing Machinery.

[LMS85] Leslie Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *J. ACM*, 32(1):52âĂŞ78, January 1985.

[MH13] Hammurabi Mendes and Maurice Herlihy. Multidimensional Approximate Agreement in Byzantine Asynchronous Systems. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC, June 2013.

[PBRT11] Maria Potop-Butucaru, Michel Raynal, and SÃľbastien Tixeuil. Distributed computing with mobile robots: An introductory survey. In *2013 16th International Conference on Network-Based Information Systems*, pages 318–324, Los Alamitos, CA, USA, sep 2011. IEEE Computer Society.

[SV16] Lili Su and Nitin H. Vaidya. Fault-tolerant multi-agent optimization: Optimal iterative distributed algorithms. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, page 425âĂŞ434, New York, NY, USA, 2016. Association for Computing Machinery.

[VG13] Nitin H. Vaidya and Vijay K. Garg. Byzantine Vector Consensus in Complete Graphs. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC, July 2013.

[WL88] Jennifer Lundelius Welch and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, 1988.