TD 13: Approximate Agreement, Renaming in AMP

In this TD, assume that there are n processes $p_1, \ldots, p_n$ and that the model of communication is the Asynchronous Message Passing model (AMP). Assumptions on the AMP model:

- All processes know n. If we assume that the model has at most f crash failures, the processes know f as well.
- Processes can send messages to everybody else.
- There is no upper bound on the time it takes for a message to be received, and that messages sent to non-faulty processes (processes that do not crash) are always received.

1 Approximate Agreement

Assume that at most f processes may crash. As shown during the class, consensus cannot be solved in the AMP model with even one crash failure. However, it is possible to solve a close variant of the consensus task, which is known as the *ε-approximate agreement* task. Any algorithm solving this task must satisfy the following conditions:

Processes start with real input values in $[0, 1]$, and must output values in $[0, 1]$, satisfying

- **ε-Agreement:** If non-faulty processes p_i and p_j decide v_i and v_j, then $|v_i - v_j| \leq \varepsilon$.
- **Validity:** If a non-faulty process (i.e., a process that does not crash) outputs v, then v must belong to the convex hull of all the processes inputs (even those of faulty ones), i.e., there exist processes with inputs c_1 and c_2 such that $c_1 \leq v \leq c_2$.
- **Termination:** All non-faulty processes decide.

The goal is to design an algorithm that solves ε-approximate agreement resilient to at most f crash failures.

Definition. (Averaging function) Define function avg_f so that, given a set $V = \{v_1, \ldots, v_N\}$ where $v_1 \leq \cdots \leq v_N$,

$$\text{avg}_f(V) = \frac{v_1 + v_{f+1} + \cdots + v_{(\lambda-1)f+1}}{\lambda}$$

where $\lambda = \lceil N/f \rceil$.

† **Problem 1.1.** Suppose V, W, and U are nonempty multisets with $|V| = |W| = m$, $V \subseteq U$, $W \subseteq U$, and $|V - W| = |W - V| \leq f$. Show that

$$\left|\text{avg}_f(V) - \text{avg}_f(W)\right| \leq \frac{\max(U) - \min(U)}{\lceil m/f \rceil}.$$

Hint: Show that $\max(v_{if+1}, w_{if+1}) \leq \min(v_{(i+1)f+1}, w_{(i+1)f+1})$ where $V = \{v_1 \leq \cdots \leq v_m\}$ and $W = \{w_1 \leq \cdots \leq w_m\}$.

Solution. We first prove the statement in the hint. We show that $v_{if+1} \leq w_{(i+1)f+1}$ (the case $w_{if+1} \leq v_{(i+1)f+1}$ is proved similarly). Assume for the sake of contradiction that $v_{if+1} > w_{(i+1)f+1}$. Then v_{if+1} is strictly larger than $(i+1)f + 1$ elements of W. Since v_{if+1} is strictly larger than at most if elements of V, it follows that there are at least $f + 1$ elements of W that are not in V, which contradicts $|W \setminus V| \leq f$. Thus for all $i < \lceil m/f \rceil - 1$, we have $\max(v_{if+1}, w_{if+1}) \leq \max(v_{(i+1)f+1}, w_{(i+1)f+1})$.

Letting $\lambda = \lceil m/f \rceil$, it follows that

$$\begin{aligned}
\left|\mathrm{avg}_f(V) - \mathrm{avg}_f(W)\right| &= \frac{1}{\lambda}\left|\sum_{i=0}^{\lambda-1} v_{if+1} - \sum_{i=0}^{\lambda-1} w_{if+1}\right| \\
&\leq \frac{1}{\lambda}\sum_{i=0}^{\lambda-1} |v_{if+1} - w_{if+1}| \\
&= \frac{1}{\lambda}\sum_{i=0}^{\lambda-1} \max(v_{if+1}, w_{if+1}) - \min(v_{if+1}, w_{if+1}) \\
&= \frac{1}{\lambda}\left[\max(v_{(\lambda-1)f+1}, w_{(\lambda-1)f+1}) - \min(v_1, w_1)\right] \qquad \text{(From the hint)} \\
&\leq \frac{\max(U) - \min(U)}{\lambda}.
\end{aligned}$$

□

† **Problem 1.2.** Let U be the set of initial values of all the processes. Give an algorithm $\mathcal{A}$ which satisfies the following: If U' is the set of output values of the non-faulty processes given by $\mathcal{A}$, then

$$\max(U') - \min(U') \leq (\max(U) - \min(U)) \left\lceil \frac{n-f}{f} \right\rceil^{-1},$$

and moreover, $[\min(U'), \max(U')] \subseteq [\min(U), \max(U)]$.

Proof. Consider the following simple algorithm.

Algorithm for process p_i with input v_i

1. Send v_i to all processes (including itself)
2. Wait to receive $n - f$ messages, and assimilate all the received values into the multiset V_i.
3. Output $\mathrm{avg}(V_i)$

We first note that all non-faulty processes must decide a value since they all send their values and there are at least $n - f$ of them.

Let p_i and p_j be two non-faulty processes. Since $|V_i \cap V_j| + |V_i \cup V_j| = |V_i| + |V_j|$ and $|V_i \cup V_j| \leq n$, we have $|V_i \cap V_j| \geq 2(n-f) - n = n - 2f$, from which it follows that $|V_i \setminus V_j| \leq (n-f) - (n-2f) = f$, and similarly $|V_j \setminus V_i| \leq f$. We can apply the result of the previous problem and obtain that for all $i \neq j$,

$$|\mathrm{avg}(V_i) - \mathrm{avg}(V_j)| \leq (\max(U) - \min(U)) \left\lceil \frac{n-f}{f} \right\rceil^{-1},$$

from which it follows that

$$\max(U') - \min(U') \leq (\max(U) - \min(U)) \left\lceil \frac{n-f}{f} \right\rceil^{-1}.$$

□

Problem 1.3. Derive an ε-approximation algorithm from $\mathcal{A}$. How large is f with respect to n?

Solution. Let $D = \lceil (n-f)/f \rceil$ and let $S = \log_D 1/\varepsilon$. Note that we need $n > 2f$. We slightly modify algorithm $\mathcal{A}$ as follows.

Algorithm for process p_i with input v_i

Let v_i be the initial value of p_i
For $r = 1, \ldots, S$:
1. Send $\langle r, v_i \rangle$ to all processes (including itself)
2. Wait to receive $n - f$ messages of the form $\langle r, * \rangle$, and assimilate all the second components into the multiset V_i.
3. $v_i \leftarrow \text{avg}(V_i)$

Output v_i

The proof of ε-agreement and validity (outputs of deciding processes belonging to convex hull of all process inputs) should be easy enough. □

2 Renaming Task

Surprisingly, even though consensus is not solvable in the AMP model, there is a task which is the complete opposite of the consensus task and is solvable:

Definition 2.1 (Renaming task)**.** The renaming task with *initial name space* of size M (one can have $|M| = \infty$) and *new name space* of finite size N and $M > N$ is defined as follows.

Every process initially has as input a *distinct* identifier from the initial name space. Any algorithm solving the renaming task must guarantee that:
- Processes that decide must all output distinct identifiers.
- All non-faulty processes must decide.

Note: Even though we shall refer to the processes as $p_1, \ldots, p_n$, the processes themselves are not aware of this indexing, otherwise one would have a trivial algorithm where p_i outputs i.

Consider the following algorithm where processes have distinct initial identifiers from an *unbounded* domain. We also assume that $f < n/2$ where f is the maximum possible number of crash failures.

Algorithm at process p_i with initial identifier id_i

0. Initialize set $known \leftarrow \{id_i\}$ and counter $c \leftarrow 0$
1. a. Send $known$ to every other process
   b. $c \leftarrow 1$
2. Wait until you receive a message V':
   a. If $V' \subsetneq known$ then goto 2
   b. If $V' \setminus known \neq \emptyset$ then: $known \leftarrow known \cup V'$, goto 1
   c. If $known = V'$ then:
      a. $c \leftarrow c + 1$
      b. If $c < n - f$ then goto 2 else goto 3
3. a. $v \leftarrow |known|$
   b. Let r be the rank of id_i in the (sorted) set $known$
   c. Choose your new name to be the pair $\langle v, r \rangle$.
4. Continue forever; whenever you receive a message V' such that $V' \setminus known \neq \emptyset$:
   a. $known \leftarrow known \cup V'$
   b. Send $known$ to every other processor

Problem 2.2. Try to understand what the algorithm is doing, in particular at steps 2 and 3.

† **Problem 2.3.** The processes are clearly exchanging certain sets. A set V is *stable* if there is some process p_i that has received $n - f - 1$ messages containing identical copies of V while the value of its *known* variable is also V.

Show for $f < n/2$ that the set of stable sets is totally ordered (i.e., every stable set is contained in some other stable set) in any run of the algorithm.

Solution. Let V and V' be two stable sets that occur at two different processes p_i and p_j. By the definition of a stable set, V must have been broadcast by at least $n - f$ processes, and same for V'. Since $2(n - f) > n$, there is at least one process that had broadcast both V and V' at some points during the run. We see from the algorithm that the set *known* at a process can only grow or remain unchanged. This means that either $V \subseteq V'$ or $V' \subseteq V$. □

†† **Problem 2.4.** Show for $f < n/2$ that every non-faulty processor eventually obtains a stable set V. Conclude that all such processors decide.

Solution. Consider some non-faulty process p_i. At some point the value of *known* at p_i will stop growing (as it is bounded above by the set of initial values). Let $final$ be the final value of *known* at p_i. Consider the point at which the value of *known* became $final$ in step 2b. At that point p_i would have broadcast $final$ to all the other processes.

All non-faulty processes (at least $n - f$ of them) receive $final$. Let p_j be such a process, and let V be the value of its *known* variable when it receives $final$ from p_i. Since p_j would have broadcast V after step 2b and since p_i stops at $final$, we have $V \subseteq final$. Thus when p_j receives $final$, it changes its *known* to $final$ in either step 2b or 4, and rebroadcasts $final$, which eventually reaches p_i. Thus p_i receives the message with value $final$ from all other non-faulty processes. This shows that every non-faulty process obtains a stable set. □

† **Problem 2.5.** Show for $f < n/2$ that the new names chosen by the non-faulty processes are distinct. What is the size of the new name space?

Solution. Let p_i and p_j be two processes which decide a new name. Suppose V_i and V_j are the stable sets which they use. If $|V_i| \neq |V_j|$, clearly the new name is different. If $|V_i| = |V_j|$, since the stable sets in a run are totally ordered, we must have $V_i = V_j$. Thus the ranks of id_i and id_j (the initial names of p_i and p_j) must be different, implying that p_i and p_j still choose different names. □

††† **Problem 2.6.** Show that for any $N \geq n + 1$, if $f \geq n/2$, then there is no algorithm for solving the renaming task with new name space of size N. *Note:* Even having an intuition for this is enough.