1 Coding for Interactive Communication

Suppose we have a scenario where two people, Alice and Bob, exchange messages in some alphabet \( \Sigma \) over a noisy channel. Let \( N = \alpha n \) be the number of characters are exchanged through \( n \) messages, each with some overhead \( \alpha > 1 \). Suppose an adversary can corrupt \( \varepsilon N \) characters, for some fraction \( \varepsilon \). Note that if a single message is corrupted, all following conversations will be affected, since in a conversation future exchanges are dependent on all messages exchanged so far.

2 Algorithm for Large Alphabet

Assume \( \log |\Sigma| = \Theta(\log n) \), where we are sending symbols from \( \Sigma \). In this algorithm we are Alice; Bob will perform analogous actions.

- Have conversation for \( B \) steps for a fixed \( B = 1/\sqrt{\varepsilon} \) (we will later see why)
- \( R_A \) = random seed generated by Alice
- Send \((R_A, H_A = H_{R_A}(T_A), \ell_A = |T_A|)\), where \( T_A \) is transcript of messages so far for Alice.
- Receive \((\tilde{R}_B, \tilde{H}_B, \tilde{\ell}_B)\) (these may be noisy, also applied on same transcript from Bob)
- If \( H_{\tilde{R}_B}(T_A) = \tilde{H}_B \), continue
- Else, backtrack \( B \) steps in \( T_A \), if \( |T_A| \geq \tilde{\ell}_B \), backtrack \( B \) more. (This helps reverse a single bad error that we overlook, so we don’t get stuck).

In this algorithm, each party exchanges a conversation block of size \( B \), and then exchange a small constant number of verification characters, which include a random seed \( R_A \), a hash of the transcript (into some logarithmic sized space), and the length of the transcript (for synchronization). Depending on whether the verification matches, either party may backtrack, which is where the conversation is no longer synchronized. Luckily, we will see that the adversary can only cause a limited amount of desynchronization.

**Observation 1** Assuming no hash collisions, above algorithm requires at most \( n/B + 32\varepsilon B \) iterations until termination.

**Proof** The sending of the correct messages will take \( n/B \) iterations, since we need to send \( n \) messages and each iteration exchanges \( B \) of them. Then, note that each single corruption introduced by the adversary will only cause the above algorithm to backtrack a constant number of iterations, therefore we get the above bound. ■

**Theorem 2** Assuming no hash collisions, this algorithm has communication blowup \( \alpha = 1 + \Theta(1/\sqrt{\varepsilon}) \).
To minimize the number of iterations we want to balance the two quantities \( n/B \) and \( 32 \times \varepsilon \times B \), which so we pick \( B = \frac{1}{\varepsilon} \) to balance cost of errors for large \( B \) and the proportion of overhead for small \( B \). Then, since each iteration we send \( B \) messages and 2 characters for the hashed value and length (which each require \( \theta(\log n) \) bits, therefore each can be represented by a constant number of characters in the alphabet) the total characters we need to send using this scheme is therefore

\[
N = (B + 2)(n/B + 32\varepsilon n) = n(1 + \Theta(\varepsilon B + \frac{1}{B})) = n(1 + \Theta(1/\sqrt{\varepsilon}))
\]
as desired.

3 Correctness and Analysis

We will define a potential Function \( \Phi \) which measures progress made in the above algorithm.

\[
\Phi = \ell^+ - \ell_1^- - \ell_2^-
\]

where we define \( \ell^+ \) as the longest consecutive number of successful iterations from the start (i.e. number of iterations where the transcript of Alice matches that of Bob), and \( \ell_1^- \) is the number of iterations that Alice has surpassed this point in her transcript. Define \( \ell_2^- \) similarly for Bob. More formally, let \( T_A \) and \( T_B \) be Alice and Bob’s transcripts, respectively. Then we can define

\[
\ell^+ = \left\lfloor \frac{1}{B} \left| \{j : T_A[1,j] = T_B[1,j]\} \right| \right\rfloor
\]

\[
\ell_1^- = \left( \frac{|T_A|}{B} - \ell^+ \right)
\]

\[
\ell_2^- = \left( \frac{|T_B|}{B} - \ell^+ \right)
\]

Claim 3 In any iteration, \( \Phi \) decreases by at most 3.

Proof This follows because in any iteration, Alice (similarly Bob) either progresses or backtracks by at most one block. Thus, \( \ell^+ \) decreases by at most 1 and \( \ell_1^- + \ell_2^- \) increases by at most 2.

Claim 4 In any iteration, if there is no corruption or hash collisions in a round \( \Phi \) increases by at least 1.

Proof This follows because in any iteration, if \( T_A = T_B \), both progress by a block, changing only \( \ell^+ \) by +1. If \( T_A \neq T_B \), then either \( |T_A| = |T_B| \) and \( \ell_1^- + \ell_2^- \) decreases by 2 and \( \ell^+ \) increases by at most 1, due to both backtracking, or \( |T_A| \neq |T_B| \), and only \( \ell_1^- + \ell_2^- \) changes by -1 due to only the person with higher block number backtracking.

Theorem 5 Accounting for hash collisions, the algorithm still only has communication blowup \( \alpha = 1 + \Theta(1/\sqrt{\varepsilon}) \).

Proof We can bound \((1 + \sqrt{\varepsilon}) < 2\), so accounting for hash collisions, we have that \( \Phi_{final} = \frac{n}{B} + 32\varepsilon n - (2n\varepsilon + h_c) \times 7 \), where \( h_c \) is the number of hash collisions, since each collision may cause a constant number of additional iterations, and we further know that the blowup will be at most a factor of 2. We can see that since the hash function size is \( \Theta(\log n) \), using the Chernoff bound, with high probability there are no more than \( 4\varepsilon n \) collisions (suppose since each hash collision occurs with probability \( 1/\log n \)). Thus we see \( \Phi_{final} \geq n/B \), therefore \( \ell^+ \geq n/B \) and so we must have successfully and correctly exchanged all conversation after \( n/B + 32\varepsilon B \) iterations of the algorithm, which is the same as the previous bound given without hash collisions.
4 Extension to a Binary Alphabet

The above method uses a rather large alphabet to keep blowup of each exchange small, as only constant verification characters are required. Now, we will extend this algorithm to a binary alphabet.

4.1 Hashing Modification

**Definition 6** Suppose we have bitstrings $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$. Then for any bitvector $r \in \{0, 1\}^n$, define the inner product hash $h_r(x) = \langle x, r \rangle$ and similarly for $y$. Note that the probability of collision is at most $\frac{1}{2}$, and we can repeat the process by generating a random $r$ log $n$ times to get high probability of success.

Now, let $s = \log(\frac{1}{\varepsilon})$ be the hash size. Since the above hash is a linear test (we take a dot product), we can instead use a $\delta$-biased seed without affecting the outcome by much. Thus now, we instead require log $n + s$ total bits of randomness for a $\delta$-biased space. So in the message sent, the hash output has length log $n + s$ bits, and length of message is log $n$. We can also protect randomness with an error correcting code with some (small) overhead.

However, we can no longer send the lengths of the transcripts since they will require $\Theta(\log n)$ bits. Instead of length checking, we instead maintain a parameter $t$ which represents how far we may be willing to backtrack. Intuitively, the farther we get, the more we may need to backtrack farther back because we no longer send exact lengths and it may take a while to reverse multiple rounds. Thus, in each iteration we increment the parameter $t$, and round the length to the nearest $2^t$, send the hash of transcripts from there. Thus, when we backtrack, we will always return to that nearest power of 2. The analysis for this algorithm is similar, but requires a more complex potential function (which is omitted). For a detailed description see [Hae14].

**Acknowledgements** These notes are based on previous scribe notes from 2014 by Vijay Bhat-tiprolu.

**References**