Resource-Competitive Error Correction

Varsha Dani*
Department of Computer Science
University of New Mexico
Albuquerque, NM, USA
varsha@cs.unm.edu

ABSTRACT

We present a resource-competitive Monte Carlo algorithm for the problem of error correction for message transmission along a noisy channel when a limited amount of feedback is available. To transmit a message of length n in the presence of $T \leq n/\log n$ adversarial errors, our algorithm sends $n + 2\sqrt{n(T+1)\log n} + cT\log n$ bits.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Data communications*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms, Theory

Keywords

Resource-Competitive Analysis; Noisy Channel; Reliable Communication

1. INTRODUCTION

In classical theoretical computer science we are used to thinking of an adversary who is all-powerful and can take actions that maximally disrupt our algorithms, at no cost or trouble to himself. While this is a useful construct when our goal is to design algorithms with provable guarantees for every possible input, it has been pointed out (see, *e.g.*, [4]) that there are many situations in distributed computing where this view of the adversary is overly pessimistic. In particular, for problems on networks involving many nodes it is reasonable to suppose that *bad* nodes, which try to disrupt the computation, are subject to the same resource constraints as the good nodes which follow the prescribed protocol.

*This material is based upon work supported by the National Science Foundation under Grant No. CCF-1320994.

FOMC'14, August 11, 2014, Philadelphia, PA, USA.

ACM 978-1-4503-2984-2/14/08 ...$15.00.
http://dx.doi.org/10.1145/2634274.2634280.

To model this, Gilbert, King, Saia and Young [4] defined a notion of *resource-competitiveness*. Informally, an algorithm is resource competitive if the cost to the algorithm is bounded by a function of the cost to the adversary. We raise a minor criticism of their definition, and then present a resource-competitive algorithm for the problem of error correction for message transmission along a noisy channel when a limited amount of feedback is available.

2. RESOURCE-COMPETITIVE ANALYSIS

We now review the definition of resource-competitiveness from [4]. Let G be the set of good players and F the set of adversarially controlled (faulty) players in the network; these sets are not known up front and membership in them may be determined adversarially and sometimes adaptively. Let $\mathcal{A}$ denote the prescribed algorithm, and let $C(\alpha, j)$ denote the cost to player j over an execution α of $\mathcal{A}$. Note that for good players this is the cost for following $\mathcal{A}$, whereas for faulty players it is the cost of whatever strategy they have followed.

Definition 2.1. Let $g(.)$ and $a(.)$ be functions that take as input the set of costs for the good and faulty players, respectively. Let $T = a(\{C(\alpha, j)\}_{p_j \in F})$. Then, an algorithm $\mathcal{A}$ is (ρ, g, a, τ)-*resource-competitive* if

$$g(\{C(\alpha, j)\}_{p_j \in G}) \leq \rho(T) + \tau$$

for any execution α.

Here, the functions g and a allow us to switch between the aggregate cost to all players or the average cost to the players or the maximum cost to any player, or whatever is appropriate in the particular problem being tackled. The function ρ is a function of T and possibly other problem-related parameters, while $\tau > 0$ may depend on other problem-related parameters, but *not* on T, and represents the cost to the algorithm when there is no attack. Obviously it is desirable that ρ and τ be as small as possible.

One slight drawback of this definition is that it fails to take into account the relationship between T and other problem parameters, if any. On the face of it, it would seem that the ability to design an algorithm with a ρ that is sublinear in T is better than one in which ρ is linear in T, but except in the case where T is asymptotically a completely free parameter, this is misleading. If T is in any way constrained by the other parameters, then the dependence of ρ on T alone may not capture the true cost of the algorithm. See Theorem 4.1 and Remark 4.2 for or [5] and the remarks about it in Section 3 for concrete examples.

Another issue is with τ. Gilbert *et al.* [4] define τ as the cost of the algorithm "when there is no attack". However, this does not distinguish between the underlying cost of the problem at hand in an attack-free environment (*i.e.*, the *promise* of no attack) and the cost of the algorithm when there is a possibility of attack, but in fact no attack occurs. In the latter case, the algorithm must incur some cost for testing for attacks, or risk terminating incorrectly. For example, sending a message over a noise-free channel costs the length of the message; if there is a possibility of noise on the channel, then it is not enough to simply transmit the message. Additional bits must be sent to ensure that the correct message is received, and this is necessary even if it turns out afterwards that there was no interference. It seems preferable to separate this cost from the underlying cost of the attack-free protocol, as it is interesting to consider what is the minimum cost to deal with the mere *threat* of attack.

3. RELATED WORK

Resource-competitive algorithms have been studied in the domain of wireless sensor networks. Here, the devices running the protocols have limited battery life and it is important that the devices following the protocol do not die before completing their task, even if some devices are actively trying to disrupt them.

King, Saia and Young [6] studied a model in which Alice wants to send a message to Bob over a communication channel that may be jammed by an adversary. The entire message may be transmitted during a single time slot at unit cost to Alice. However if Bob is not listening (for unit cost) in that time slot or the adversary jams the time slot (again for unit cost) then Bob does not receive the message. King, Saia and Young presented a Las Vegas algorithm for Alice to transmit and Bob to listen so that if the adversary jams T time slots Alice and Bob can guarantee receipt of the message with an expected total cost of $O(T^{\varphi-1}+1)$ where φ is the golden mean.

Gilbert and Young [5] studied communication under a different attack model in which a single authenticated node Alice wants to send a message to n good nodes over a single channel. The adversary controls $\Theta(n)$ Byzantine nodes that may try to arbitrarily disrupt the communication. In particular they may jam the channel or attempt to spoof the correct nodes and request retransmissions from Alice. For an arbitrarily fixed parameter $k \geq 2$, each player has a budget of $\tilde{O}\left(n^{1/k}\right)$. Gilbert and Young present a Monte Carlo algorithm such that for any $\varepsilon > 0$, with high probability at least a $1-\varepsilon$ fraction of the good nodes receive the message while ensuring that if the adversary spends $T = \tilde{O}(n^{1+1/k})$ to disrupt, then each good node incurs a cost of no more than $\tilde{O}\left(T^{\frac{1}{k+1}}+1\right)$. Note that T is constrained due to individual budget constraints on the Byzantine nodes and the resulting cost guarantee to the good players is the same order of magnitude as their individual budgets.

Gilbert, King, Pettie, Porat, Saia and Young [3] also consider a model in which Alice is trying to send a message to many nodes, in the presence of a jamming adversary with an unknown (but finite) budget. Their adversary is less powerful, in that it cannot spoof the recipients. Against this adversary, Gilbert *et al.* [3] present a Monte Carlo algorithm that achieves an expected per node cost of $O(\sqrt{T/n}\log^4 T + \log^6 n)$ when the adversary's cost is T.

The definition of resource-competitiveness does not require the good players to spend less than the adversary. Nonetheless, this is the case in all of the aforementioned resource-competitive algorithms, at least asymptotically. In contrast, in our work the good players must inherently pay more than the adversary. This is mitigated by the fact that our cost model counts the actual number of bits transmitted, whereas in the aforementioned works, only the number of messages sent was counted, regardless of their size.

Gilbert, Guerraoui, and Newport [2] also study the problem of Alice and Bob communicating in the presence of a jamming adversary, and show matching upper and lower bounds indicating that an adversary who can jam T slots can delay successful communication by $2T + \log|V|/2$ rounds, where V is the message space. Here the adversary appears to have the advantage. However these results are not directly comparable to the resource-competitive algorithms mentioned above, because the metric they are trying to optimize is delay, rather than transmission cost, and in in particular, the players need not be incurring much cost while they wait out the delay guaranteed by the lower bound.

4. PROBLEM STATEMENT AND RESULT

Honest parties Alice and Bob want to communicate. Alice has an n-bit message that she wants to send to Bob. However, her communication channel to Bob is vulnerable to an adversary who can flip bits on the channel. If it were known how many bits the adversary would flip, Alice could use an error-correcting code with appropriately chosen parameters to ensure receipt of the correct message; see, e.g., [8]. This, however, costs bandwidth inversely proportional to the rate of the code, and is therefore a constant factor blowup over transmitting the message in the clear. Thus, if an adversary causes Alice to use error correction, he has already caused damage, even if no bits are actually flipped on the channel!

Now suppose Alice has a limited ability to periodically receive some feedback from Bob. For instance, suppose Alice breaks her message up into smaller blocks, and Bob can send fingerprints of the blocks he has received. The transmission cost for this feedback is charged to Alice (or alternatively we can think of charging the cost of both Alice and Bob's transmissions to the algorithm.) Thus, it is not efficient for Bob to do something like echoing the message he heard. In this situation, is it possible for Alice and Bob to tailor their transmissions to the amount of interference actually observed on the channel, ideally sending only about as many bits as if they had known in advance how many bits would be flipped? In other words, can we come up with a resource-competitive algorithm?

If the bit flips on the channel are adversarial, then the answer is no. The reason is that if the adversary can flip bits in Alice's as well as Bob's transmissions, then he can change Alice's message and also change Bob's fingerprint, so that the fingerprint matches the changed message. Then neither Alice nor Bob can detect that the message Bob received is not the same as the message Alice sent. This is sometimes called a "man in the middle" attack. To avoid this, we will make the simplifying assumption that the fingerprints that Bob sends are not corruptible (*i.e.*, the channel from Bob to Alice is noise-free) and only messages from Alice are subject to adversarial bit flips. An alternative assumption could be that both channels are noisy, but that they are *private*, so that the adversary can flip bits on the channels, but must

do so without actually seeing the traffic on them. We note that the assumption of private channels does not reduce the power of the adversary to random noise. The adversary still knows the algorithm used by Alice and Bob, and can flip arbitrary patterns of bits in an attempt to thwart it. In particular, the adversary can do things like flipping exactly one out of every B bits, or flipping very long sequences of consecutive bits. Such patterns seldom occur with i.i.d. random noise. In practice, private channels can easily be implemented using cryptography.

Under either of these assumptions, we present an algorithm for Alice and Bob with the following guarantee:

THEOREM 4.1. *Let $T = O(n)$ be the number of bits flipped by the adversary. Then the total number of bits sent by Alice and Bob is always $O(n)$ and when $T \leq n/\log n$ it is no more than*

$$n + 2\sqrt{2c(T+1)n\log n} + cT\log n. \quad (1)$$

With high probability, Bob recovers the correct n-bit message from Alice.

To put this in the context of Definition 2.1, here g is the total number of bits sent by Alice and Bob, $\rho(T) = 2\sqrt{2c(T+1)n\log n} - 2\sqrt{2cn\log n} + cT\log n$ and τ is either $n + 2\sqrt{2cn\log n}$ or just $2\sqrt{2cn\log n}$ depending on whether one wants to include the cost n of sending the message on a clear channel in τ or not. Thus, our algorithm is resource-competitive.

Section 5 is devoted to describing and analyzing our algorithm, and thereby proving the theorem. Note that we have assumed adversarial noise. If the noise is actually i.i.d. the algorithm (and proof) will work without the additional assumption of private channels or noise-free fingerprints, even without the noise rate being known.

REMARK 4.2. The term $2\sqrt{2c(T+1)n\log n}$ in (1) dominates the term $cT\log n$ when $T = O(n/\log n)$, which ties in with our earlier remarks on the dependence of the function ρ from Definition 2.1 on other problem parameters.

The fingerprints sent by Bob come from the following

THEOREM 4.3 (NAOR AND NAOR [9], ALON ET AL. [1]). *There exists a constant $q > 0$ and an ensemble of hash families $\{H_k\}_{k\in\mathbb{N}}$ such that for every $k \in \mathbb{N}$ and for every $h \in H_k$, $h : \{0,1\}^{\leq 2^k} \to \{0,1\}^{qk}$ is poly-time computable, it is efficient to sample $h \leftarrow H_k$ using only qk random bits, and for all $x \neq y \in \{0,1\}^{\leq 2^k}$ it holds that*

$$\operatorname*{Prob}_{h\leftarrow H_k}[h(x) = h(y)] \leq 2^{-k}.$$

This implies that a fingerprint of size $c\log n$ using (and including) such a random hash function applied to strings of length less than n has a polynomially small probability to incorrectly match. The error probability of the algorithm comes from taking a union bound over the failure probabilities of all the fingerprints sent.

Finally, we note that Theorem 4.1 is asymptotically fairly close to optimal. A lower bound of $\Omega\left(\sqrt{nT(\log n + 1 - \log T)}\right)$ is implied by the work of Kol and Raz [7].

5. INTUITION, ALGORITHM AND COST ANALYSIS

As a preliminary attempt, let us try something very simple-minded. Suppose Alice splits her message into n/B blocks of size B. To each block, she prepends $\log n$ bits representing the block number. She sends Bob a block (headed with its number) and waits for the fingerprint of what Bob received. If the fingerprint matches the fingerprint applied to the block she sent, then Alice moves on to the next block. Otherwise she resends the same block. Of course this approach is not guaranteed to terminate at all, since the adversary can flip one bit per transmission and Alice would never get past the first block. However, as mentioned in the remarks in Section 4, this can be fixed by giving up and using an error correcting code after some threshold of bit flips.

Not surprisingly, this does pretty badly. There is a tension between wanting to set the block sizes small, so that Alice is not spending too much on resends when errors are detected on the one hand, and wanting to set the block sizes large in order to minimize the cost of the fingerprints on the other. Since the block size is B, this costs $(n/B)c\log n$ for fingerprints, and $B + (c+1)\log n$ per error thereafter. This is ruinously expensive if there is a very large number of errors. But as long as $B = \omega(\log n)$, if there are no errors, it costs only $o(n)$ overhead. In fact, in this case, the $O(B)$ cost of the resends also does not start to hurt until there are $\Omega(n/B)$ flips. With this in mind, let us refine the algorithm, so that the resend cost associated with errors depends non-linearly on the number of errors. We should be willing to pay aggressively for error correction in the initial stages, and only back off when our total expense so far is fairly high. More precisely, for $j \geq 0$ let $\rho(n,j)$ denote the cost we are willing to pay for detecting/correcting the jth bit flip. Let $\rho(n,0)$ be the overhead cost of admitting the possibility of errors (this was previously called $\tau(n)$). Let T denote the actual number of bits flipped by the adversary. Then the (maximum) cost of our algorithm is $n + C(n,T)$, where

$$C(n,T) = \sum_{j=0}^{T} \rho(n,j)$$

and the property we hope our algorithm will satisfy is

$$C(n,T) = \begin{cases} o(n) & \text{when } T = o(n) \\ O(n) & \text{when } T = \Theta(n) \end{cases}$$

How do we decide how to set the initial block size? Here again, a reasonable intuition is that we should be willing to pay about as much for the first flip as we do for overheads. We are ready to present a first version of the algorithm.

Algorithm, version 1

Let $B_0 = \sqrt{n\log n}$ denote the initial block size. Alice begins the algorithm in phase 0, by sending a block of the first B_0 bits of her message prefixed with $\log n$ 0s representing the index of the start of the message. She then receives a fingerprint from Bob. If the fingerprint does not match what she sent, she increments her count of the number of errors and resends the block with the same prefix. If it matches, she proceeds to the next block of size B_0. Each time she sends a block, she prefixes it with $\log n$ bits representing the index of the starting bit for the block. For $i \geq 1$, if the number of errors Alice has seen reaches $4^i - 1$ then Alice

moves into phase i of the algorithm. In phase i, the block size is $B_i = \frac{B_{i-1}}{2}$. At each step, Alice sends Bob the first B_i bits of her message that have not previously been successfully transmitted together with the index of their starting location.

We note that the block size in phase i is

$$B_i = \frac{B_0}{2^i} = \frac{\sqrt{n \log n}}{2^i}$$

One caveat: since the fingerprints are of size $\log n$, if the block size ever becomes that small, Alice will be paying at least a constant blowup from then on. So at that point she gives up and sends the entire message using an error correcting code and incurs a constant blowup. Say that for some fixed $\varepsilon > 0$ she gives up when the block size is $\log^{1+\varepsilon} n$. This means that the last phase before she gives up is i such that

$$\frac{\sqrt{n \log n}}{2^i} = \log^{1+\varepsilon} n$$

In other words,

$$i = \frac{1}{2} \log n - \left(\frac{1}{2} + \varepsilon\right) \log \log n$$

The error threshold for this phase is

$$4^{i+1} = \frac{4n}{\log^{1+2\varepsilon} n}$$

Let's analyze the cost of this algorithm. Let $c \log n$ be the overhead per block, *i.e.*, the cost of sending the starting index of the block, and the cost of receiving the fingerprint.

If there are no bit flips then the algorithm reaches the end of the simulation in phase 0, and we have only paid $c \log n$ overhead each for n/B_0 blocks, so $\rho(n, 0) = c\sqrt{n \log n}$. Now suppose there are some bit flips. What do we pay for the jth bit flip?

If $4^i \le j < 4^{i+1}$ then we are in phase i and the block size is B_i, so to resend the block in which the jth error occurred, we pay $B_i + c \log n$, where the $c \log n$ is for the extra start index and fingerprint. Note that $c \log n = o(B_i)$.

However, there is also an indirect cost for errors, once we are past phase 0. This comes from the fact that once the block size is reduced, we are sending more start indices and fingerprints than we originally accounted for as overhead. Originally, we were sending n/B_0 blocks and we assigned the entire cost of these to $\rho(n, 0)$. However, once we move into a new phase, the block size halves, so for the remainder of the protocol we must send twice as many blocks and incur the corresponding overhead costs. Moreover, note that all the errors could be occurring in repeated resends of the very first block, so at any time what remains of the protocol could be length n, *i.e.* the whole protocol. Thus, as we move into phase $i+1$, we are committing to send up to n/B_{i+1} blocks, which is n/B_i more than the n/B_i we already committed previously. The cost of these n/B_i additional blocks is

$$\frac{n}{B_i} c \log n = \frac{2^i c n \log n}{\sqrt{n \log n}} = 2^i c \sqrt{n \log n}\ .$$

Let's assign these to phase i.

Now recall that the total cost of phase i from resends was

$$\begin{aligned}(4^{i+1} - 4^i)(B_i + \log n) &= 3 \times 4^i \left(\frac{\sqrt{n \log n}}{2^i} + c \log n\right) \\ &= (3 + o(1)) 2^i \sqrt{n \log n}\end{aligned}$$

Thus, the additional $2^i \sqrt{n \log n}$ that we assigned to phase i for overhead is of the same magnitude as the resend costs in phase i, and the total cost attributed to phase i, which we'll denote $\phi(n, i)$, is $(3 + c + o(1))\, 2^i \sqrt{n \log n}$. Amortizing this total cost over all the errors in phase i, we have, for $4^i \le j < 4^{i+1}$

$$\rho(n, j) = (1 + c/3 + o(1))\, B_i \sim (1 + c/3 + o(1))\, \frac{\sqrt{n \log n}}{2^i}$$

Now what about the total cost when there are T errors? If $T = 0$ we pay $\rho(n, 0) = c\sqrt{n \log n}$. Otherwise if $4^i \le T < 4^{i+1} \le \frac{4n}{\log^{1+2\varepsilon} n}$ then the algorithm ends in phase i. In this case, we pay

$$\begin{aligned}C(n, T) &= \sum_{j=0}^{T} \rho(n, t) \\ &\le c\sqrt{n \log n} + \sum_{j=0}^{i} \phi(n, j) \\ &\le c\sqrt{n \log n} + (3 + c + o(1)) \sqrt{n \log n} \sum_{j=0}^{i} 2^j \\ &\le (3 + c) \sqrt{n \log n} \left(2^{i+1}\right) \\ &\le (6 + 2c) \sqrt{T n \log n}\end{aligned}$$

where the last inequality follows since $4^i \le T$ implies $2^i < \sqrt{T}$. Thus, we've shown that for $T \le \frac{4n}{\log^{1+2\varepsilon}}$,

$$C(n, T) = (6 + 2c) \sqrt{T n \log n}$$

which is $o(n)$. For larger T, we give up and use an error correcting code, so in that case $C(n, T) = \Theta(n)$.

We note that here we have not optimized the block size to get the best constant instead of $6 + 2c$. We will do that in the next section, where we present a smoothed version of the algorithm.

Remark 5.1. The property that $C(n, T) = o(n)$ only goes up to $T = \frac{4n}{\log^{1+2\varepsilon}} < \frac{4n}{\log n}$, and does not hold for all T which are $o(n)$. By choosing ε to be suitably small, we can make it work arbitrarily close to $\frac{4n}{\log n}$ but beyond that we must pay $\Theta(n)$. This seems currently unavoidable, because the fingerprint size is order $\log n$. It may be possible to get around this by using smaller fingerprints, but this is not completely straightforward, since then the failure rate of the fingerprints is too large for our union bound.

Algorithm, version 2 (smoothed)

In this section we present a smoothed version of the same algorithm.

Let $c \log n$ be the overhead cost per block sent, *i.e.*, the combined cost of the block number and the fingerprint. Let B_0 denote the initial block size. We'll start with $B_0 = \gamma\sqrt{n \log n}$ for some γ to be determined. The block size will change every time an error is encountered. For $1 \le j \le n/\log n$, let B_j denote the new block size after the jth error. We'll set

$$B_j = B_0 \left(\sqrt{j+1} - \sqrt{j}\right) = \frac{B_0}{\sqrt{j+1} + \sqrt{j}}$$

Thus, at any given time, Alice and Bob simulate the noise-free protocol in blocks of size B_j where j is the number

of errors encountered so far. Alice initiates each block by sending Bob the block number. At the end of each block, Bob sends a fingerprint of the transcript. If the fingerprint matches then Alice signals starting the next block (by sending its number). If it does not match, she signals repeating the block (by sending the same block number). If a block is being repeated, Alice and Bob both update their block size by incrementing j by 1. If j gets to be $n/\log n$, Alice and Bob give up on fingerprints and Alice sends the message with an error correcting code..

Let's analyze the cost of this algorithm. If there are no errors, then the entire protocol is simulated in blocks of size B_0, each accompanied by a block number and fingerprint of total size $c \log n$. Thus the overhead is

$$\rho(n, 0) = \frac{cn \log n}{B_0} = \frac{c}{\gamma}\sqrt{n \log n}.$$

Now suppose $j \geq 1$. What do we pay for the jth error? There are two sources of cost associated with the jth error. One source is the wasted transmission of the block in which the error was detected (which has already been paid) and the other is the increased overhead due to the decrease in block size.

The block in which the jth error is detected is of size B_{j-1}, since there were $j-1$ previous errors. Thus the cost of the wasted transmission is

$$\begin{aligned} B_{j-1} + c \log n &= B_0 \left(\sqrt{j} - \sqrt{j-1}\right) + c \log n \\ &= \gamma\sqrt{n \log n}\left(\sqrt{j} - \sqrt{j-1}\right) + c \log n \end{aligned}$$

where the $c \log n$ term is for the block number and fingerprint.

The block size is now changed to B_j. The overhead due to numbering and fingerprinting with every block of size B_j (assuming that the algorithm will end while sending blocks of this size) is

$$\frac{cn \log n}{B_j} = \frac{c}{\gamma}\sqrt{n \log n}\left(\sqrt{j+1} + \sqrt{j}\right)$$

However, part of this has already been charged to previous errors. Indeed, $\frac{cn \log n}{B_{j-1}}$ of overhead was already charged to errors $1, \ldots j-1$. Thus, the overhead charged to the jth error is

$$\begin{aligned} &\frac{cn \log n}{B_j} - \frac{cn \log n}{B_{j-1}} \\ &\quad = \frac{c}{\gamma}\sqrt{n \log n}\left[\left(\sqrt{j+1} + \sqrt{j}\right) - \left(\sqrt{j} + \sqrt{j-1}\right)\right] \\ &\quad = \frac{c}{\gamma}\sqrt{n \log n}\left(\sqrt{j+1} - \sqrt{j-1}\right) \end{aligned}$$

Putting these together, we have

$$\begin{aligned} &\rho(n, j) \\ &= \sqrt{n \log n}\left[\frac{c}{\gamma}\sqrt{j+1} + \gamma\sqrt{j} - (c/\gamma + \gamma)\sqrt{j-1}\right] + c \log n \end{aligned}$$

Finally, the total cost when there are T errors is

$$\begin{aligned} &C(n, T) \\ &= \sum_{j=0}^{T} \rho(n, j) \\ &= \sqrt{n \log n}\left(\frac{c}{\gamma} + \sum_{j=1}^{T}\left(\frac{c}{\gamma}\sqrt{j+1} + \gamma\sqrt{j} - (c/\gamma + \gamma)\sqrt{j-1}\right)\right) \\ &\quad + \sum_{j=1}^{T} c \log n \\ &= \sqrt{n \log n}\left(\frac{c}{\gamma}\sqrt{T+1} + (c/\gamma + \gamma)\sqrt{T}\right) + cT \log n \\ &\leq \left(\frac{2c}{\gamma} + \gamma\right)\sqrt{(T+1)n \log n} + cT \log n \end{aligned}$$

This is optimized by setting $\gamma = \sqrt{2c}$ whereupon we get

$$C(n, T) \leq 2\sqrt{2c(T+1)n \log n} + cT \log n\,.$$

This completes the cost analysis analysis of the presented algorithm, and proves (1). To complete the proof of Theorem 4.1 we only need to argue that Bob receives the correct message with high probability. However this follows easily from Theorem 4 and a union bound, since the only way that Bob fails to get the correct message is if one or more of the fingerprints match incorrectly.

6. CONCLUSION

We have presented a resource-competitive Monte Carlo algorithm for the problem of error correction in message transmission when some feedback is available. It is hoped that the ideas used herein may be applicable to designing resource-competitive algorithms for multi party computation.

Acknowledgements

We thank Tom Hayes, Mahnush Movahedi, Jared Saia, Maxwell Young and the anonymous reviewers for helpful comments.

7. REFERENCES

[1] N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures and Algorithms,* 3(3):289-304, 1992.

[2] S. Gilbert, R. Guerraoui, and C. Newport. Of Malicious Motes and Suspicious Sensors: On the Efficiency of Malicious Interference in Wireless Networks. *OPODIS 2006.*

[3] S. Gilbert, V. King, S. Pettie, E. Porat, J. Saia, and M. Young (Near) Optimal Resource-Competitive Broadcast with Jamming. *SPAA 2014*

[4] S. Gilbert, V. King, J. Saia, and M. Young Resource-Competitive Analysis: A New Perspective on Attack-Resistant Distributed Computing. *FOMC 2012.*

[5] S. Gilbert and M. Young. Making Evildoers Pay: Resource-Competitive Broadcast in Sensor Networks. *PODC 2012*

[6] V. King, J. Saia, and M. Young Conflict on a Communication Channel. *PODC 2011*

[7] G. Kol and R. Raz. Interactive channel capacity. *STOC 2013.*

[8] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes.* North-Holland: New York, NY, 1977.

[9] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.,* **22**(4):838-856, 1993.