1 High-rate locally-correctable and locally-testable codes with sub-polynomial query complexity

Abstract:

Locally-correctable codes (LCCs) and locally-testable codes (LTCs) are special families of error-correcting codes that admit extremely efficient sub-linear time algorithms for error correction and detection, respectively, while making a few queries to the received word. These codes were originally studied in complexity theory because of their close relationship to program checking and probabilistically-checkable proofs (PCPs), but subsequently they were extensively studied for their own sake.

In this work, we construct the first locally-correctable codes and locally-testable codes with constant rate, constant relative distance, and sub-polynomial query complexity and running time. Specifically, we show that there exist binary LCCs and LTCs with block length n, constant rate (which can even be taken arbitrarily close to 1), constant relative distance, and query complexity and running time $\exp(\sqrt{\log n})$. Previously such codes were known to exist only with query complexity and running time n^β (for constant $\beta > 0$), and there were several, quite different, constructions known.

Along the way, we also construct LCCs and LTCs over large (but constant) size alphabets, with the same query complexity and running time $\exp(\sqrt{\log n})$, which additionally have the property of approaching the Singleton bound: they have almost the best-possible relationship between their rate and distance. This has the surprising consequence that asking for a large alphabet error-correcting code to further be an LCC or LTC does not require any sacrifice in terms of rate and distance! Such a result was previously not known for any sub-linear query complexity.

If time permits I will also discuss a very recent work that substantially improves the query complexity of LTCs to quasi-polylogarithmic function of the form $(\log n)^{O(\log \log n)}$.

Joint work with Swastik Kopparty, Or Meir and Shubhangi Saraf

An error-correcting code is a map $C : \Sigma^k \to \Sigma^n$. Σ is the alphabet, k is the message length, and n is the codeword length.

Two basic parameters are the rate $\frac{k}{n}$ and the distance

$$\min_{u,v \in C} \{d(u,v) := \mathbb{P}_{i \in [n]}(u_i \neq v_i)\}$$

It equals twice the proportion of errors the code can correct (picture).

There are 3 operations associated with codes.

- Encoding: compute map C.
- Testing/detecting errors: test $u \in C$.
- Correcting: find $c \in C$ nearest to u.

We're interested in sub-linear time algorithms that do these operations. We want them to run in time $\ll n$. They can't even make 1 pass over the codeword! We need several assumptions.

- Random access: they can access each coordinate with some cost.
- Probabilistic (has a small chance of failure).

The first family is locally testable codes (LTC's), which have sublinear time testing algorithms.

Definition 1.1: A **local tester** is given as input $u \in \Sigma^n$ and outputs accept/reject. It satisfies the following:

$$u \in C \implies \mathbb{P}(\text{accept}) = 1$$
$$u \notin C \implies \mathbb{P}(\text{reject}) \geq 0.99 d(u, C).$$

The rejection should depend on the distance, because if it just differs in one location, you probably won't find it.

Locally correctable codes are codes with sublinear time correcting algorithms.

Definition 1.2: A **local corrector** is given as input $u \in \Sigma^n$ such that $d(u, C) < \frac{1}{2}d(C)$ and $i \in [n]$ and is required to output the ith coordinate of codeword closest to u with probability ≥ 0.99.

Definition 1.3: A **local decoder** is given as input $u \in \Sigma^n$ such that $d(u, C) < \frac{1}{2}d(C)$ and $i \in [k]$ and is required to output the ith coordinate of the message m such that $C(m)$ is the codeword closest to u with probability ≥ 0.99.

LDC's seem more natural, but LCC's imply LDC's. If you can decode every coordinate of the codeword, you can decode every coordinate of the message.

Fact 1.4: For linear codes, LCC's imply LDC's.

Our main goal is to minimize the number of queries and the running time.

1.1 Previous work and motivations

Most of the research has focused on a constant number of queries and constant distance. The goal is to maximize the rate. People studied this regime because it would be powerful; sometimes you can do amazing stuff with 2–3 queries (PCP). Studying this regime has applications in complexity theory. They are connected to property testing, hardness of approximation, etc.

In practice, all known constructions of such codes have vanishing rate: as $n \to 0$, rate goes to 0. This is very bad for applications in data transmission and storage.

For LTC's,

- the best constructions have rate $\frac{1}{\text{polylog}\, n}$, due to Ben Sannon-Sudan 2005, Dinur 2007, Meir 2009, and Videman 2013.

- It is not known if we can achieve a constant rate. This is the biggest open problem in this area.

There is an implicit connection to PCPs. Every time someone gets a better LTC, with the same techniques people get a better PCP, and vice versa. However there is no formal connection.

For LCC's,

- we cannot get constant rate (Katz-Trevisan 2000).
- Reed-Muller codes give $\exp(-n^{\varepsilon})$, and there are LDC's with rate $\exp(-\exp(\sqrt{\ln n \ln\ln n}))$ by Yekhanin 2007 and Efremenko 2009.

1.2 High-rate regime

We are interested in constant rate and constant distance, and we want to minimize the number of queries. Ex. $\log n$ queries is fine.

The practical motivation is to encode massive data.

1. Solution 1: break into chunks and encode each part separately.

   Pro: The recovery time is small. We only have to read bits that are close.

   Con: in the presence of errors, the failure probability is the number of pieces times $\exp(-|\text{small}|)$.

2. Encode the message in one big chunk.

   Pro: Failure probability is $\exp(-|\text{big}|)$.

   Con: The recovery time is —big—.

What is known? Up to a few years ago, the only known examples were Reed-Muller. For all $\varepsilon > 0$, you can have a code with number of queries is n^{ε} and the rate is $\exp(-\frac{1}{\varepsilon})$. This was believed to be best possible.

Actually you can do better. For every $\varepsilon > 0$, you can have a code with queries n^{ε} and rate $1 - \varepsilon$.

Here are known constructions.

- The first codes were multiplicity codes (LCC's), generalization of Reed-Muller codes. In Reed-Muller codes you evaluate polynomials; here you evaluate polynomials and their derivatives (Kopparty-Saraf-Yekhanin 2011).

  Algebraic flavor.

- Tensor codes (LTC's): BenSasson-Sudan 2006, Videman 2011.

  Combinatorial flavor.

- Lifted Reed-Solomon codes (both LTC's and LCC's), due to Guo-Kopparty-Sudan 2013.
- Expander codes (LCCs), due to Hemenway-Ostrovsky-Wooters 2013.

  Graph-theoretic.

1.3 Our results

We show that we can improve the number of queries to be subpolynomial. We give high-rate LTCs/LCCs with subpolynomial number of queries

Theorem 1.5: For every $\varepsilon > 0$ there is a LCC and a LTC with number of queries equal to $\exp(\sqrt{\ln n \ln \ln n})$ and rate $1 - \varepsilon$.

Theorem 1.6: For all $\varepsilon > 0$ there is a LTC with the number of queries $(\ln n)^{O(\ln \ln n)} = \exp((\ln \ln n)^2)$.

Additionally, we get a code which approaches the singleton bound with distance $\sim \varepsilon$.

1.4 Proof overview

1. Component 1: Get high-rate LTC's/LCC's with subpolynomial queries with subconstant distance.
2. Component 2: Apply distance amplification (due to Alon-Luby 1996).

- For the first part, to get LTCs with $\exp(\sqrt{\ln n})$ queries, we use tensor codes of high dimension.
- For LCCs with $\exp(\sqrt{\ln n})$ queries, we use multiplicity codes of high dimension.
- For LTCs with $(\ln n)^{O(\ln \ln n)}$ queries, use an iterative construction. Alternate tensor product and distance amplification.

This is similar in spirit to techniques/results in complexity theory:

- Zigzag product (Reingold-Vadhan-Wigderson)
- $SL = L$ (Reingold)
- Gap amplification (Dinur)

Rather than just use tensor products, alternate tensor product and amplification.

We use ideas from combinatorial constructions of LTC's due to Meir 2009.

Component 2 is based off of Alon-Luby 1996, based on expanders. Given a LTC/LCC with the following parameters:

- distance $\delta' = o(1)$,
- rate r,
- number of queries q,

for every δ, ε, we obtain a code with the following parameters

- distance $\delta' = \Omega(n)$, $\delta \gg \delta'$,
- rate $r(1 - \delta - \varepsilon)$,
- number of queries $q \operatorname{poly}(\frac{1}{\delta'}, \frac{1}{\varepsilon})$.

Previously they used this to increase the distance. We actually amplify distance of local codes from subconstant to constant.

The application is more to classical data transmission and storage. For appications to complexity, we usually want a constant number of queries.