Rank-Metric Codes in Sage

Arpit Merchant

March 25, 2016

Abstract

In this project, our goal is to extend the Coding Theory library in Sage by implementing base classes for linear rank-metric codes and Gabidulin codes. We also aim to add functionality for encoding and decoding algorithms and methods to answer basic queries about these codes while ensuring efficient operations.

Contents

1 Introduction **2**

2 About Me **2**
2.1 Contact Details 2
2.2 My Background 2
2.3 Related Contributions 3

3 Project Details **4**
3.1 Modules 4
3.2 Schedule 6
3.3 Deliverables 7
3.4 Risk Management 8

4 Other Engagements **8**

1 Introduction

Sage is a viable open source mathematics software package that offers an extensive toolbox for algebra, combinatorics and numerical computations. It has great support for Coding Theory with various families such as Generalized Reed-Solomon Codes, BCH Codes, Cyclic codes (among others) which are already available, along with encoding and decoding algorithms and methods to answer basic queries about these codes.

Rank-metric codes are rapidly gaining traction in the research community due to their wide applicability in random network coding, cryptography and distributed systems. The aim of this project is to extend Sage by creating base functionality for representing linear rank-metric codes and answering basic questions about them. In particular, this also includes implementation of Gabidulin codes as well as decoding algorithm(s) for it. Using the in-built implementations for fields and polynomials directly, might prove to be slow or insufficient and therefore a parallel aim is enhance these representations to make the critical operations more efficient.

2 About Me

2.1 Contact Details

- Name: Arpit Merchant
- Email: arpitdm@gmail.com
- Homepage: https://researchweb.iiit.ac.in/~arpit.merchant/
- My handle on Github, Skype, Git-trac and everywhere else is `arpitdm`

2.2 My Background

I am a Masters student at IIIT-H, India. I completed my undergraduate in Computer Science last year. I have been a Teaching Assistant for the Discrete Mathematics and Optimization Methods courses in the past. I had a course on Linear Algebra that covered all the basics of groups, fields, rings, polynomials, vector spaces. I also took an elective course on Error Correcting Codes which included topics such as Binary Symmetric Channel, Hamming/Singleton/Gilbert-Varshamov Bounds, Linear/Block/Reed-Solomon/Cyclic Codes, ML/Standard-Array decoding. And so I'm well versed with the basics of coding theory.

In a research project on pseudorandomness, I studied the properties and efficient constructions of pseudorandom objects such as expander graphs and list decodable codes, generators as well as basic derandomization techniques. The aim of the project was to design probability distributions that are statistically close to the uniform and can be distinguished by polynomials with only a small probability of error, but are generated using fewer random bits. Thus I have experience when it comes to reading research papers and mathematical analysis (as this project could require).

On the technical side, I've been programming in C and Python for nearly five years now, so I'm quite comfortable with both. I've undertaken and completed major systems projects in the past such as:

- **Compiler for the Decaf programming language**
  - Built a syntax analyzer using *flex* and *bison* and modified it so it generates an abstract syntax tree (AST) representation of the source program written in Decaf.
  - Designed a module that converts the AST into an LLVM intermediate representation and implemented an optimization pass manager.
- **Hadoop Distributed File System**
  - Implemented a distributed file system with two major components namely NameNode and DataNode, that supports all file operations, block mechanisms and allocations. It handles communication through RPC and provides 2-way replication.
  - Developed a MapReduce program for processing the data in the file system by allowing for scalability, fault tolerance and optimized execution through marshalling.

I haven't used Sage much in the past, but I've started using it a lot more over the past few weeks in preparation for this proposal.

2.3 Related Contributions

In order to get myself familiarized with the codebase and development workflow, I spoke with David Lucas and Johan Nielson, the two mentors for this project for tasks related to the Coding Theory library of Sage.

- Simplify LinearCode.zero method (Ticket #20113)
  - David had already written a patch to fix this ticket but the Trac system requires that a second programmer "review" it before it can be added to Sage.

  - I worked through the Reviewer Checklist and reported back. After confirmation, I gave it a positive review and closed the ticket.

- Improving Efficiency of LinearCode.NearestNeighborDecoder method (Ticket #20201)
  - I opened the ticket based on the idea provided by David and discussed the solution idea with Johan.
  - I've submitted a commit and am working towards getting it merged.

3 Project Details

The goal of this project is to enhance existing functionality for the pre-requisite linear algebra and implement rank-metric codes. More concretely, this project will involve the following modules.

3.1 Modules

- **Module 1: Faster Operations in Different Representations of Fields**
  Rank-metric codes (and Gabidulin in particular) require frequent switching between 'small' and 'big' fields, $GF(q)^m$ and $GF(q^m)$. When q is not prime, the current Sage implementation could become slow or it could be insufficient or both. David Lucas (one of the mentors for this project) wrote Ticket #20039 for such conversions. But it has not yet been reviewed and is thus not yet part of Sage. Moreover, there could be further modifications required to this ticket. For example, the Frobenius operation (computing a^q given a in $GF(q^m)$) is used extensively in rank-metric codes and support for this needs to be added.

- **Module 2: Review of Skew-Polynomial tickets**
  Linearized polynomials form a non-commutative ring and are a special case of Skew Polynomials. When the derivation is zero and the Frobenius automorphism is used, Skew Polynomials become Linearized Polynomials. Gabidulin codes are based on the latter. However Ticket #13215, written by Xavier Caruso adds support in Sage for Skew Polynomials. This massive ticket has not been reviewed and could require modifications in order to be used in the implementation of Gabidulin codes.

- **Module 3: Base Class for Rank Metric Codes**
  The original incarnation of the coding theory library was a collection of methods that implemented combinatorial properties of various types of linear codes such as BCH,

Cyclic, Golay, etc. as can be seen here. But with ACTIS, classical code families have been written as classes (eg Generalized Reed-Solomon Codes). This feature proposes the implementation of a similar base class for Rank Metric codes. This includes:

  - Basic properties of the code - Minimum Distance, Covering Radius, Generator Matrix, Parity Check Matrix, etc. (as per the definition of the Abstract Linear Code class and as required)
  - Encoding and Decoding algorithm(s)

- **Module 4: Base Class for Gabidulin Codes**
  Gabidulin codes, analogs of Reed-Solomon codes in the rank-distance metric, can be defined as evaluation codes of degree-restricted linearized polynomials. More formally, *A linear Gabidulin code* `Gab[n, k]` over $\mathbb{F}_{q^m}$ *of length* $n \leq m$ *and dimension* $k \leq n$ *is the set of all the words, which are the evaluation of a q-degree-restricted linearized polynomial* $f(x) \in \mathbb{L}_{q^m}[x]$:

$$\text{Gab[n, k]} = \left\{(f(g_0)f(g_1)...f(g_{n-1})) = f(\mathbf{g}) : \deg_q f(x) < k\right\}$$

  where the fixed elements $g_0, g_1, ..., g_{n-1} \in \mathbb{F}_{q^m}$ *are linearly independent over* $\mathbb{F}_q$.
  We propose their implementation in Sage along with:

  - Basic combinatorial properties such as Minimum Rank Distance, Parity Check Matrix, Generator Matrix, etc (as required)
  - Encoding algorithm
  - The Syndrome decoding algorithm for it

- **Module 5: Testing and Documentation**
  I plan on writing the tests and documentation for every commit after it is successfully added into Sage before moving on to the next task. This would involve:

  - Use the unittest python module to test the proper working of the implemented features
  - Perform doctest and code coverage analysis if and when possible
  - Speedup checks to make sure that the new code is not completely slowing down Sage
  - End-to-end testing to ensure the proper integration of the code with the rest of Sage. One way would be to compare final outputs with those from the unit tests
  - Develop the required documentation (design, usage, theory and README)

- **Module 6: Error Erasure Cases** *[if time permits]*
  Section 5.3 of the paper titled *Coding for Errors and Erasures in Random Network Coding* describes a new decoding algorithm that can handle complex error erasure cases and can drastically speed up the process. If time permits, this can be implemented.

3.2 Schedule

- **April 22 - May 22 (Community Bonding Period):** I will spend this time getting more familiar with the coding theory library. I will explicitly study and understand code family class structure (grs.py, code_construction.py, linear_code.py) in order to have clarity about the new style of implementation. I will also use this time to study in detail the theory behind skew-polynomials, extension fields and normal basis, as well as rank-metric and Gabidulin codes.

- **May 23 - June 27 (Pre-mid term evaluation)**

  - May 23 - June 6: Implement Module 1 (*estimated 2 weeks*)
    * Read the ticket and create a UML-diagram of the code. Community discussion. We'll understand how switching between $GF(q^m)$ and $GF(q)^m$ has been implemented. It will be good to have statistics. (3 days)
    * Based on the UML, discuss with the community the advantages and pitfalls of the ticket. Decide on Frobenius and any other operations that need to be sped up. Increase efficiency of operations in $GF(q^m)$ representations when q is not prime. (3 days)
    * Implement the modifications and commit them. (6 days)
    * Write a thorough test framework and rewrite documentation (3 days)
  - June 7 - June 27: Implement Module 2 (*estimated 3 weeks*)
    * Read the ticket and create a UML-diagram of the code. We'll thus know what in skew polynomials has been implemented already and what is missing. Community discussion.(5 days)
    * Based on the UML, thoroughly review the ticket. Decide what needs to be polished and how. Open tickets accordingly. Community discussion. (4 days)
    * Implement the modifications and commit them. (5 days)
    * Write a thorough test framework. Write any required documentation. (3 days)
    * Buffer period. Perform code review to ensure that the design and efficiency are up to scratch. Finish any unfinished work.

- **June 28 - August 15 (Post-mid term evaluation)**
  - June 28 - July 19: Implement Module 3 (*estimated 3 weeks*)
    * Open discussion with community for design, actual needs, feature requests. Decide on which encoders and decoders are to be used for Rank-Metric codes and which combinatorial properties to implement. Potential scalability issues. (5 days)
    * Implement the base class for Rank-metric codes and the properties as decided above. (5 days)
    * Implement at least one encoding and decoding algorithm. (5 days)
    * Perform preliminary testing to ensure correctness and safety. Review integration of all modules implemented so far.
    * Write the necessary documentation and the rest of the time is buffer period.
  - July 20 - August 14: Implement Module 4 (*estimated 3 weeks*)
    * Implement class for Gabidulin codes. (4 days)
    * Implement encoder(s) and decoder(s) for it. Ongoing discussions. (4 days)
    * Implement methods to answer combinatorial queries for these codes. (4 days)
    * Write unittests for this module and thorough end-to-end tests for the entire project. The documentation for this module will be written as and when commits happen. (7 days)
    * Discussion of overall design, interface questions, verification of efficiency. Make changes to improve on this if and as required. (7 days)
- **August 15 - August 23 (Final Week)**
  - Finalize documentation.
  - Do one final code review.
  - Integrate all the components and finish any unfinished work.

3.3 Deliverables

- Summary of Ticket #20039
- Edit and Append code to Ticket #20039
- Summary and Review of Ticket #13215

- Base class for Rank-metric codes
- Base class for Gabidulin codes
- Syndrome Decoding for Gabidulin codes
- Tests
- Documentation

3.4 Risk Management

The primary risk that we anticipate is the modifications and enhancements to the skew-polynomials and field representations. Since these affect the entire Sage package, discussions can become large scale and community-wide. This can cause the implementation process to slow down. The idea is to identify those methods necessary for this project and narrow the focus to only those.

4 Other Engagements

I am not participating in any other internships during the duration of the GSOC programme. I will be writing my the thesis for my Masters degree till mid-June perhaps (though it could take a little longer). However, that will require only about 15 hours a week and so it will not take away from this project. Towards July-end/early-August, I might be relocating to a new college and that could take a couple of days. Apart from these, I don't have any other commitments.

Acknowledgments

David Lucas and Johan Nielson, the mentors for this project have been extremely helpful. The ideas in this proposal, the work on the patches and the planning could not have been possible without their help.