CS 687 / CM 730: Project Topics Due: Thu Apr 6, 2023

CS 687 / CM 730 students are required to write a project on an aspect of symbolic computation or computer algebra. This will typically consist of reviewing a particular topic (generally 2 or 3 related papers or textbook chapter) and possibly doing an implementation. The written part of the project will typically be about 10 pages long and should be properly structured and supported by appropriate citations of all materials. There should be many concrete examples to illustrate the concepts. Papers listed here should be available through Google, but send me a message if you have trouble finding them. They should only be thought of as a starting point for your investigations. Below are some possible project topics and starting references. But please, propose any avenue of investigation you wish, and come and discuss this with me. Some of you already have indicated a topic for your project.

1. **Further examination of sparse linear algebra.** For sparse matrices, especially over finite fields, standard methods of matrix arithmetic are too slow. A sparse matrix is one with very few non-zero entries. Such matrices arise over finite fields in applications such as factoring integers. Given an $n \times n$ matrix A over a finite field $\mathbb{Z}/(p)$, standard linear algebra requires $O(n^3)$ operations in $\mathbb{Z}/(p)$ to solve a system, even when most entries in A are zero. Wiedemann in 1986 introduced a new class of "sparse" algorithms which require $O(n\mu + n^2)$ operations, when A has μ nonzero elements. If A has only, say $20n$ nonzero entries (out of a possible n^2), then this algorithm requires only $O(n^2)$ operations, a big improvement over Gaussian elimination. Investigate Wiedemanns and related algorithms, described in the papers below. Consider problems related to system solving such as computing the rank and minimal polynomial, and also for dealing with singular matrices and possibly matrices over fields with coefficient growth like $\mathbb{Q}$. Implement Wiedemanns or a related algorithm in Maple and try to beat Maples built-in routines for very sparse matrices (this will not be easy). Here are some papers:

   - D. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Transactions on Information Theory, IT-32, pp. 56–62, 1986.
     This paper is a classic, but somewhat difficult to read.
   - E Kaltofen and B. D. Saunders, *On Wiedemanns Method of Solving Sparse Linear Systems*, Proc. AAECC-9, Springer Lecture Notes in Comp. Sci., v. 539, pp. 29–38, 1991.
     This is much more accessible, looks at singular matrices, and matrices over $\mathbb{Q}$ as well.
   - J. von zur Gathen and J. Gerhard, *Modern Computer Algebra* (i.e., the course text), Sections 12.3 and 12.4.
   - V. Shoup, *A Computational Introduction to Number Theory and Algebra*, v. 2.2. Chapter 18.
     This is a very nice text, available free online at `http://www.shoup.net/ntb/`

2. **Polynomial decomposition.** Polynomial decomposition seeks to represent a polynomial $f \in \mathsf{F}[x]$ (for a field F) as a functional composition $f(x) = g(h(x))$. For example:

$$x^6 + 6x^4 + 2x^3 + 9x^2 + 6x + 2 = g(h(x)) \text{ for } g = x^2 + 1 \text{ and } h = x^3 + 3^x + 1.$$

Algorithms for finding such g and h go back to Zippel in 1976, while the theory of decompositions goes back at least to Ritt in 1922. You should investigate the computational problem of decomposition of polynomials. This material is actually quite accessible, and uses some of the algorithms developed in class. You can also consider the decomposition of rational functions (see below). Papers to consider are:

- D. Kozen and S. Landau, *Polynomial decomposition algorithms*. J. Symb. Comput., 7:445–456, 1989.
  This is the first polynomial-time algorithm for polynomial decomposition.
- Von zur Gathen, *Functional decomposition of polynomials: the tame case.* J. Symb. Comput., 9:281–299, 1990.
  Shows that polynomial decomposition can essentially be done in linear time, using Newton iteration over the polynomials, as discussed in class.
- C. Alonso, J. Gutierrez, and T. Recio, *A rational function decomposition algorithm by near-separated polynomials.* J. Symbolic Computation, 19:527–544, 1995.
  This is an easy to implement (exponential time) algorithm. It would be nice to have a Maple implementation.

3. **Iterated Frobenius factoring.** Section 14.7 of the text discusses the so-called "Iterated Frobenius" method for factoring polynomials over finite fields, which is faster (at least in terms of analysis) than the Cantor-Zassenhaus algorithm. Investigate this algorithm and implement it in Maple (or C++ using NTL). Compare it with the built in algorithm, and other algorithms (for example, Berlekamps algorithm, which is presented in Section 14.8). There has been a recent development on the complexity of factoring in STOC 2008. Summarize the contribution in this work (it is difficult to read, but the final results are quite digestible), and the overall state of factoring polynomials over finite fields.

- J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, Section 14.9 and 14.10.
- J. von zur Gathen and V. Shoup, *Computing Frobenius maps and factoring polynomials*, Computational Complexity v.2., pp.187–224, 1992.
- C. Umans, *Fast polynomial factorization and modular composition in small character* , ACM STOC 2008.
- E. Kaltofen and V. Shoup, *Subquadratic-time factoring of polynomials over finite fields*, Mathematics of Computation, v. 67(223), pp. 1179–1197, 1998.

4. **Primality testing.** Chapter 18 of the text presents an overview of efficient methods to determine if a number is prime (up to, but not including recent deterministic methods). Use this as a starting point for a survey of different methods. Be sure to identify the cost and

the randomness requirements the described methods. Finish with an overview of the most recent polynomial-time deterministic algorithms (the famous recent paper is called *Primes is in P*, by Agrawal, Kayal and Saxena, though there are much more accessible descriptions available). Consider implementing one or more of the probabilistic methods.

5. **Sparse polynomials.** When polynomials are described by a list of coefficient and exponent pairs their representation can be much shorter. For example

$$20x^{5000} - 13x^{2500} + 3x + 2$$

might be represented as $\{(20,5000),(30,2500),(3,1),(2,0)\}$. Many researchers have considered algorithms for computing with such representations. Surprisingly, some basic problems like GCD are NP-hard. Take a look at

- D. A. Plaisted. Sparse complex polynomials and polynomial reducibility. J. Comp. and System Sciences, 14:210–221, 1977.
- D. A. Plaisted. New NP-hard and NP-complete polynomial and integer divisibility problems. Theor. Computer Science, 31:125–138, 1984.

In light of the above, it is perhaps even more surprising that some problem have fast algorithms. For example, we can quickly find roots of a sparse integer polynomial. See

- F. Cucker, P. Koiran and S. Smale, *A polynomial time algorithm for Diophantine equations in one variable*. J. Symbolic Comput., 27(1):21–29, 1999. ISSN 0747–7171.

Perform a survey of what can and cannot be computed with sparse polynomials (these are sometimes called lacunary or super-sparse polynomials. Consider implementing the above algorithm of Cucker, Koiran and Smale.

6. **Groebner basis.** An accessible and self-contained introduction to Groebner Bases is given in Chapter 10 of the book:

- K. O. Geddes, S. R. Czapor and G. Labahn. "Algorithms for Computer Algebra", Kluwer Academic Publishers, 1992.

The aim of this project would be to familiarize yourself with the main concepts needed to understand and implement in Maple Buchberger's algorithm to compute Groebner Bases, namely Algorithm 10.2 on page 446. More specifically, these concepts are: term orderings, reduction, s-polynomials. These are illustrated with numerous examples in Chapter 10. Once you understand these concepts well, then understanding and implementing in Maple Buchberger's algorithm should be straightforward.

Another aim of the project could be to optimize your implementation by adding the zero-reduction criteria, i.e., criteria that predict in advance which reductions of pairs of polynomials in the algorithm will result to zero, so that these reductions are not performed at al.

Reductions are very time consuming and statistically speaking, 90% of reductions will yield a zero.

Once you have both implementations up and running, with and without zero-reduction criteria running in Maple, then you can experiment with the benchmark examples given in the book, and other benchmark examples from the literature, to find out how many reductions to zero are performed, for each benchmark example. You can use the reduction routine in Maple as a black box, i.e. you don't have to implement it from scratch.