Efficient Computation with Sparse and Dense Polynomials

by

Daniel Steven Roche

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Computations with polynomials are at the heart of any computer algebra system and also have many applications in engineering, coding theory, and cryptography. Generally speaking, the low-level polynomial computations of interest can be classified as arithmetic operations, algebraic computations, and inverse symbolic problems. New algorithms are presented in all these areas which improve on the state of the art in both theoretical and practical performance.

Traditionally, polynomials may be represented in a computer in one of two ways: as a "dense" array of all possible coefficients up to the polynomial's degree, or as a "sparse" list of coefficient-exponent tuples. In the latter case, zero terms are not explicitly written, giving a potentially more compact representation.

In the area of arithmetic operations, new algorithms are presented for the multiplication of dense polynomials. These have the same asymptotic time cost of the fastest existing approaches, but reduce the intermediate storage required from linear in the size of the input to a constant amount. Two different algorithms for so-called "adaptive" multiplication are also presented which effectively provide a gradient between existing sparse and dense algorithms, giving a large improvement in many cases while never performing significantly worse than the best existing approaches.

Algebraic computations on sparse polynomials are considered as well. The first known polynomial-time algorithm to detect when a sparse polynomial is a perfect power is presented, along with two different approaches to computing the perfect power factorization.

Inverse symbolic problems are those for which the challenge is to compute a symbolic mathematical representation of a program or "black box". First, new algorithms are presented which improve the complexity of interpolation for sparse polynomials with coefficients in finite fields or approximate complex numbers. Second, the first polynomial-time algorithm for the more general problem of sparsest-shift interpolation is presented.

The practical performance of all these algorithms is demonstrated with implementations in a high-performance library and compared to existing software and previous techniques.

Acknowledgements

There are many people to praise (or blame) for the quality of this work. First and foremost is my wife, Leah, who agreed to come with me to Canada and has tolerated countless long nights, meaningless scribble, and boring conversations for almost five years. Despite this document's completion, these habits of mine are likely to persist, and my hope is that her patience for them will as well.

I have always been very lucky to have the support of my family, and in particular my parents, Duane and Sheryl, deserve credit for always encouraging my academic pursuits. They managed to instill in me the importance of education without squelching creativity or the joy of learning, and I thank them for it.

A great debt is owed to my many excellent primary, secondary, and post-secondary school teachers and professors for encouraging me to ask interesting questions. I especially thank Dave Saunders at the University of Delaware for introducing me to the wonderful world of computer algebra in the summer of 2004.

In Waterloo, my academic pursuits have been moderated by healthy doses of musical expression. I thank the members of orchestra@uwaterloo, the Wellington Winds, Ebytown Brass, Brass Essentials, and the Guelph Symphony Orchestra for so many evenings and weekends of great playing.

Conversation with my fellow students in the Symbolic Computation Group lab ranges from polynomials to politics and is always enlightening. A particular thanks goes to my three office mates over the years, Scott Cowan, Curtis Bright, and Somit Gupta, as well as Reinhold Burger, Mustafa Elsheik, Myung Sub Kim, Scott MacLean, Jason Peasgood, Nam Pham, and Hrushikesh Tilak.

I have been fortunate to discuss the topics of this thesis with numerous top researchers in the world, whose valuable insights have been instrumental in various ways: Jacques Carette, James Davenport, Jean-Guillaume Dumas, Richard Fateman, Joachim von zur Gathen, Jürgen Gerhard, Pascal Giorgi, Jeremy Johnson, Pascal Koiran, George Labahn, Wen-Shin Lee, Marc Moreno Maza, John May, Michael Monagan, Roman Pearce, Clément Pernet, Erik Postma, Éric Schost, Igor Shparlinski, and Stephen Watt.

David Harvey of New York University was kind enough to email me with some ideas a few years ago which we developed together, yielding the main results presented here in Chapter 3. I thank and admire David for his great patience in working with me.

The members of my thesis examining committee all provided great feedback on this document, as well as some lively discussion. Thanks to Erich Kaltofen, Kevin Hare, Ian Munro, and Jeff Shallit.

The greatest thanks for this thesis goes to my two supervisors, Mark Giesbrecht and Arne Storjohann. Our countless conversations over the last five years have covered not only the content and direction of my research, but also teaching, publishing, and how to be a university faculty member. Their generosity in time, ideas, and funding have truly allowed me to flourish as a graduate student.

Finally, I am very grateful to the generous financial support during the course of my studies from the David R. Cheriton School of Computer Science, the Mathematics of Information Technology and Complex Systems (MITACS) research network, and the Natural Sciences and Engineering Research Council of Canada (NSERC).

Table of Contents

List of Algorithms **xi**

List of Tables **xiii**

List of Figures **xv**

1 Introduction **1**
1.1 Overview . . . 2
1.2 Polynomials and representations . . . 4
1.3 Computational model . . . 9
1.4 Warm-up: $O(n \log n)$ multiplication . . . 16

2 Algorithm implementations in a software library **21**
2.1 Existing software . . . 21
2.2 Goals of the MVP library . . . 23
2.3 Library design and architecture . . . 24
2.4 Algorithms for low-level operations . . . 27
2.5 Benchmarking of low-level operations . . . 32
2.6 Further developments . . . 34

3 In-place Truncated Fourier Transform **35**
3.1 Background . . . 35
3.2 Computing powers of ω . . . 42
3.3 Space-restricted TFT . . . 42
3.4 Space-restricted ITFT . . . 46
3.5 More detailed cost analysis . . . 47
3.6 Implementation . . . 49
3.7 Future work . . . 51

4 Multiplication without extra space **53**
4.1 Previous results . . . 53
4.2 Space-efficient Karatsuba multiplication . . . 57
4.3 Space-efficient FFT-based multiplication . . . 62
4.4 Implementation . . . 66
4.5 Conclusions . . . 68

5 Adaptive multiplication **69**

5.1 Background . . . 69
5.2 Overview of Approach . . . 70
5.3 Chunky Polynomials . . . 71
5.4 Equal-Spaced Polynomials . . . 81
5.5 Chunks with Equal Spacing . . . 87
5.6 Implementation and benchmarking . . . 89
5.7 Conclusions and Future Work . . . 91

6 Sparse perfect powers **93**
6.1 Background . . . 94
6.2 Testing for perfect powers . . . 97
6.3 Computing perfect roots . . . 106
6.4 Implementation . . . 115
6.5 Conclusions . . . 115

7 Sparse interpolation **117**
7.1 Background . . . 117
7.2 Sparse interpolation for generic fields . . . 123
7.3 Sparse interpolation over finite fields . . . 125
7.4 Approximate sparse interpolation algorithms . . . 128
7.5 Implementation results . . . 133
7.6 Conclusions . . . 135

8 Sparsest shift interpolation **137**
8.1 Background . . . 138
8.2 Computing the Sparsest Shift . . . 140
8.3 Interpolation . . . 146
8.4 Generating primes . . . 148
8.5 Complexity analysis . . . 151
8.6 Conclusions and Future Work . . . 155

Bibliography **157**

List of Algorithms

3.1 In-place truncated Fourier transform . . . 44
3.2 In-place inverse truncated Fourier transform . . . 46
4.1 Space-efficient Karatsuba multiplication . . . 58
4.2 Space-efficient FFT-based multiplication . . . 65
5.1 Chunky Multiplication . . . 73
5.2 Chunky Conversion Algorithm . . . 76
5.3 Optimal Chunk Size Computation . . . 79
5.4 Equal Spaced Multiplication . . . 83
5.5 Equal Spaced Conversion . . . 86
6.1 Perfect rth power over $\mathbb{F}_q$ . . . 99
6.2 Perfect rth power over $\mathbb{Z}$ . . . 102
6.3 Perfect power detection over $\mathbb{Z}$ . . . 104
6.4 Perfect power detection over $\mathbb{F}_q$ . . . 104
6.5 Algebraic algorithm for computing perfect roots . . . 109
6.6 Sparsity-sensitive Newton iteration to compute perfect roots . . . 111
7.1 Generic interpolation . . . 124
7.2 Verification over finite fields . . . 127
7.3 Approximate norm . . . 129
7.4 Approximate Remainder . . . 130
7.5 Adaptive diversification . . . 132
8.1 Computing the sparsest shift . . . 143
8.2 Sparse Polynomial Interpolation over $\mathbb{Q}[x]$ . . . 147

List of Tables

1.1 Instruction set for IMM model 12

2.1 Properties of machine used for benchmarking 32
2.2 Benchmarking for ten billion axpy operations modulo a word-sized prime 32
2.3 Benchmarks for sparse univariate multiplication in MVP and SDMP 33

3.1 Benchmarking results for discrete Fourier transforms 50

4.1 Values stored in D through the steps of Algorithm 4.1 59
4.2 Benchmarks versus NTL 67

5.1 Benchmarks for adaptive multiplication with varying chunkiness 90

7.1 Sparse univariate interpolation over large finite fields, with black box size ℓ, degree d, and t nonzero terms 120
7.2 Sparse multivariate interpolation over large finite fields, with black box size ℓ, n variables, degree d, and t nonzero terms 121
7.3 Finite Fields Algorithm Timings 134
7.4 Approximate Algorithm Stability 135

List of Figures

1.1 Algebraic circuit for $(-2x_1+3x_3)\cdot(x_1\cdot x_2)\cdot(5x_1-x_3)$ 5

3.1 Butterfly circuit for decimation-in-time FFT 38
3.2 Circuit for radix-2 decimation-in-time FFT of size 16 39
3.3 Circuit for forward TFT of size 11 40
3.4 Recursion tree for in-place TFT of size $n=6$ 43
3.5 Circuit for Algorithm 3.1 with size $n=11$ 44