INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9201682

Algorithms for polynomial real root isolation

Johnson, Jeremy R., Ph.D.

The Ohio State University, 1991

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

ALGORITHMS FOR POLYNOMIAL REAL ROOT ISOLATION

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By

Jeremy R. Johnson, B.S., M.S.

* * * * *

The Ohio State University

1991

Dissertation Committee:

Professor George E. Collins

Professor Wolfgang W. Kuechlin

Professor Hans Zassenhaus

Approved by

George E. Collins
Adviser

Department of Computer
and Information Science

ALGORITHMS FOR POLYNOMIAL REAL ROOT ISOLATION

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By

Jeremy R. Johnson, B.S., M.S.

* * * * *

The Ohio State University

1991

Dissertation Committee:

Professor George E. Collins

Professor Wolfgang W. Kuechlin

Professor Hans Zassenhaus

Approved by

Adviser

Department of Computer
and Information Science

© Copyright by
Jeremy R. Johnson
1991

To A.M. because the morning is so fine

ACKNOWLEDGEMENTS

I am indebted to my advisor, Professor G. E. Collins, for sharing his expertise and insight with me. This thesis owes a great deal to him, for his ideas and suggestions and more importantly for my years of apprenticeship with him. His influence will be felt for the rest of my life.

I would also like to thank the members of my committee. Professor Zassenhaus' enthusiasm and mathematical insight has been inspirational. Professor Kuechlin shared many hours of advice and discussion that has made writing this thesis much easier. Special thanks are offered to Professor Alan Woods for his last minute assistance.

I would like to thank two of my colleagues, Werner Krandick and Hoon Hong, for their encouragement and camaraderie. I would especially like to thank Werner for the use of his complex root isolation program and Hoon for the use of his partial CAD program. I would also like to acknowledge John Abbott for a discussion leading to an example favorable to the Sturm sequence algorithm.

To my parents and the rest of my family I am indebted to their constant encouragement and for providing me with an environment in which to prosper intellectually. Most of all, I would like to thank Anna for persevering with me to the completion of my thesis.

VITA

January 28, 1963 Born—New York, New York

1985 B.A. Mathematics
University of Wisconsin
Madison, Wisconsin

1988 M.S. Computer Science
University of Delaware
Newark, Delaware

1987-1991 Graduate Research Associate
Department of Computer and Information
Science
The Ohio State University
Columbus, Ohio

Publications

A report on the performance of an implementation of Strassen's algorithm, (with C.-H. Huang and R. W. Johnson), *Applied Mathematics Letters*, Vol. 4, No. 1, 1991, pp. 99-102.

A tensor product formulation of Strassen's matrix multiplication algorithm, (with C.-H. Huang and R. W. Johnson), *Applied Mathematics Letters*, Vol. 3, No. 3, 1990, pp. 67-71.

A methodology for designing, modifying, and implementing Fourier Transform Algorithms on various architectures, (with R. W. Johnson, D. Rodriguez, and R. Tolimieri), *Circuits, Systems, and Signal Processing*, Vol. 9, No. 4, 1990, pp. 449-500.

Multilinear algebra and parallel programming, (with R. W. Johnson and C.-H. Huang), In *Proc. Supercomputing '90*, IEEE Computer Society Press, Los Alamitos, California, 1990, pp. 20-31.

PARSAC-2: A multi-threaded system for symbolic and algebraic computation, (with G. E. Collins and W. Kuechlin), Technical Report, Number TR38, Department of Computer and Information Science, The Ohio State University, 1990.

Quantifier elimination and the sign variation method for real root isolation, (with G. E. Collins), In *Proc. ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, ACM Press, 1989, pp. 264-271.

On the probability of relative primality of Gaussian integers, (with G. E. Collins), In *Proc. 1988 International Symposium on Symbolic and Algebraic Computation*, Lecture Notes in Computer Science, No. 358, Springer-Verlag, New York, 1989.

Some issues in designing algebraic algorithms for the CRAY X-MP, In *Computer Algebra and Parallelism*, Ed. J. Della Dora and J. Fitch, Academic Press, 1989, pp. 179-195.

Fields of Study

Major Field: Computer and Information Science

Specializations:

Computer Algebra (Prof. G. E. Collins)

Parallel Computation (Prof. C.-H. Huang and Prof. W. W. Kuechlin)

Algebra and Computational Number Theory (Prof. H. Zassenhaus)

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
VITA	iv
LIST OF FIGURES	ix
LIST OF TABLES	xiii
CHAPTER	PAGE
I Introduction	1
II Mathematical Preliminaries	12
2.1 Notation for Computing Time Analyses	13
2.2 Polynomial Inequalities	14
2.2.1 Root bounds	17
2.2.2 Root Separation Theorems	19
2.3 Resultants, Subresultants and Polynomial Remainder Sequences . .	23
2.4 Polynomial Transformations	28
2.5 Root Counting Theorems	33
III Auxiliary Algorithms	46
3.1 Interval Operations	46
3.2 Polynomial Transformations	54
3.3 Polynomial Evaluation	60
3.4 Polynomial Root Bound Computation	65
3.5 Polynomial GCD and Greatest Squarefree Divisor Computation . .	70

IV	Real Root Isolation of Integral Polynomials	74
4.1	Sturm Sequence Based Algorithm	74
4.1.1	Computing Sturm Sequences	75
4.1.2	Isolating the Real Roots of a Polynomial Using Sturm Sequences	82
4.1.3	An Example	84
4.1.4	Recurrence Relation Evaluation of Sturm Sequences	87
4.2	An Algorithm Based on Polynomial Transformations and Descartes' Rule of Signs	90
4.2.1	The Coefficient Sign Variation Method for Real Root Isolation	94
4.2.2	The Dual Algorithm	98
4.3	An Algorithm Based on Rolle's Theorem and the Derivative Sequence	104
4.3.1	Tangent Construction	108
4.3.2	Improvement Using Fourier's Theorem	110
4.3.3	An Example	113
4.4	Computing Time Analysis	114
4.5	Empirical Behavior and Average Computing Time	121
4.5.1	Exceptional Polynomials	121
4.5.2	Random Polynomials and Average Computing Time	125
4.5.3	Polynomials with Varying Number of Real Roots	133
V	Real Algebraic Number Algorithms	143
5.1	Representation of Real Algebraic Numbers	144
5.1.1	Minimal Polynomial and Isolating Interval	144
5.1.2	Multiple Extensions	145
5.1.3	Using the Derivative Sequence	147
5.1.4	Interval Arithmetic	148
5.2	Algebraic Polynomial Operations	153
5.2.1	Permutation of Variables	156
5.2.2	Algebraic Polynomial Evaluation	162
5.2.3	Algebraic Polynomial Transformations	164
5.3	Sign Computation	167
5.3.1	Multiple Extensions	179
5.4	Algebraic Polynomial Norm	181
5.5	Algebraic Polynomial Root Bound Computation	183
5.5.1	Bounds for Real Algebraic Numbers	188

5.6	Algebraic Polynomial GCD Computation	189
5.6.1	Algebraic Polynomial GCDs and the Subresultant PRS Algorithm	192
5.6.2	Fast Test for Relatively Prime Polynomials	195
VI	Real Root Isolation of Polynomials with Real Algebraic Number Coefficients	197
6.1	Reduction to Real Root Isolation of Integral Polynomials	197
6.1.1	Computing Time	202
6.1.2	Empirical Behavior	202
6.2	An Algorithm Based on Sturm Sequences	204
6.2.1	Subresultant Sturm Sequences	209
6.3	An Algorithm Based on Rolle's theorem and the Derivative Sequence	211
6.3.1	Computing Time	213
6.3.2	Empirical Behavior	215
6.4	An Algorithm Based on Descartes' Rule and Polynomial Transformations	216
6.4.1	Computing Time	220
6.4.2	Empirical Behavior	221
6.5	Effect of the Length of the Isolating Interval	222
6.6	Use of Interval Arithmetic	223
VII	Real Root Isolation in Cylindrical Algebraic Decomposition	233
7.1	Resultants, Discriminants, and Two-Dimensional CAD	235
7.2	Ill-Conditioned Polynomials	241
	BIBLIOGRAPHY	249

LIST OF FIGURES

FIGURE		PAGE
1	IORD2 Integer, Order of 2	48
2	ILOG2 Integer Logarithm, Base 2	48
3	ITRUNC Integer Truncation	49
4	RIB Rational Interval Bisection	50
5	IPSIFI Integral Polynomial Standard Isolating Interval from Interval	52
6	IPRRS Integral Polynomial Real Root Separate	55
7	PRT Polynomial Reciprocal Transformation	56
8	IUPHT Integral Univariate Polynomial Homothetic Transformation	56
9	IUPNT Integral Univariate Polynomial Negative Transformation . .	57
10	IUPBHT Integral Univariate Polynomial Binary Homothetic Trans- formation	57
11	IUPTR Integral Univariate Polynomial Translation	59
12	RUPRE Rational Univariate Polynomial Rational Evaluation	60
13	IUPREI Integral Univariate Polynomial Rational Evaluation Integral Part	63
14	IUPRB Integral Univariate Polynomial Root Bound	68

15	IUPPRBD Integral Univariate Polynomial Positive Root Bound Using Descartes' Rule of Signs	69
16	IPGCD Integral Polynomial GCD and Cofactor	72
17	IPPSQR Integral Polynomial Pseudo-Remainder and Quotient . . .	76
18	IPNPRS Integral Polynomial Primitive Negative PRS	78
19	IPNSPRSM Integral Polynomial Negative Subresultant PRS, Modular Algorithm	81
20	IPRIST Integral Polynomial Real Root Isolation Using Sturm Sequences	83
21	IPRISS Integral Polynomial Isolating Interval Search Using Sturm Sequences	83
22	IPLEV Integral Polynomial List Evaluation of Signs	84
23	Search Tree associated with IPRISS ($S, -16, 0, 5, 3$)	87
24	IPRSEVS Integral Polynomial Recursive Sturm Sequence Evaluation of Signs	89
25	Algorithm DESCARTES	93
26	IPRICS Integral Polynomial Root Isolation, Coefficient Sign Variation Method	95
27	IPRINCS Integral Polynomial Root Isolation, Normalized Coefficient Sign Variation Method	96
28	Search Tree for IPRINCS	99
29	IPRICSD Integral Polynomial Root Isolation, Coefficient Sign Variation Method, Dual Algorithm	101

30	IPRINCSD Integral Polynomial Root Isolation, Normalized Coefficient Sign Variation Method, Dual Algorithm	102
31	Algorithm ROLLE	106
32	IPRRSTC Integral Polynomial Relative Root Separation Using Tangent Construction	111
33	ANFAF Algebraic Number from Algebraic Field Element	146
34	Interval Evaluation of the Range of a Polynomial	154
35	PTMV Polynomial Transposition of Variables	159
36	Trace of PTMV	160
37	PCPV Polynomial Cyclic Permutation of Variables	161
38	PICPV Polynomial Inverse Cyclic Permutation of Variables	162
39	IPBREI Integral Polynomial Binary Rational Evaluation, Integral Part	164
40	IPBHT Integral Polynomial Binary Homothetic Transformation . . .	165
41	IPTR Integral Polynomial Translation	166
42	AFZTEST Algebraic Field Element Zero Test	168
43	AFSIGN Algebraic Field Element Sign	169
44	AFSIGND Algebraic Field Element Sign, Using Descarte's Rule . .	172
45	AFSIGNDS Algebraic Field Element Sign at Derivative Sequence .	175
46	ATFSIGN Algebraic Tower of Fields Sign	180
47	INFAF Interval from Algebraic Field Element	188
48	AFINTMVT Algebraic Field Element Interval, Using Mean Value Theorem Interval Evaluation	190

49	AFPRP Algebraic Field Polynomial Relative Primality Test	196
50	AFPRIN Algebraic Field Polynomial Root Isolation Using the Norm	199
51	AFPRIDS Algebraic Field Polynomial Root Isolation Using the Derivative Sequence	214
52	AFPRICS Algebraic Field Polynomial Root Isolation, Coefficient Sign Variation Method	218
53	AFPRINCS Algebraic Field Polynomial Root Isolation, Normalized Coefficient Sign Variation Method	219
54	Roots of Perturbed Wilkinson Polynomial	246

LIST OF TABLES

TABLE		PAGE
1	Computing Times (in ms) of IUPHT , IUPBHT , and IUPTR . . .	61
2	Computing Times (in ms) of IUPTR , IUPTR1 , and IUPBHT . .	62
3	Empirical Computing Times (in ms) of Polynomial Evaluation	66
4	Empirical Comparison of Root Bound Computation	71
5	Computing Times (in ms) of IPGCDC	73
6	Comparison of IPLEV and IPRSEVS	90
7	IPRIST Statistics for $x^m - (ax - 1)^2$	122
8	IPRICS Statistics for $x^m - (ax - 1)^2$	123
9	IPRIST Statistics for $x^m + (ax - 1)^2$	124
10	IPRICS Statistics for $x^m + (ax - 1)^2$	124
11	Sturm Sequence Statistics for Random Polynomials	127
12	IPIISS Search Tree Statistics for Random Polynomials	128
13	IPRICS Search Tree Statistics for Random Polynomials	128
14	Comparison of Derivative Sequence Algorithms	129
15	IPRIDSF Statistics for Random Polynomials	131
16	IPRIDTCF Statistics for Random Polynomials	131

17	Sturm Sequence Statistics for Polynomials with Random Roots	134
18	IPIISS Search Tree Statistics for Polynomials with Random Roots .	135
19	IPRICS Search Tree Statistics for Polynomials with Random Roots .	135
20	Comparison of Derivative Sequence Algorithms for Polynomials with Random Roots	136
21	IPRIDSF Statistics for Random Polynomials	136
22	Timings for Chebycheff Polynomials	142
23	Empirical Computing Times (in ms) of Interval Polynomial Evaluation	155
24	Empirical Computing Times (in ms) for Permutation of Variables . .	163
25	Empirical Computing Times (in ms) for Random Algebraic Sign Com- putation	177
26	Empirical Computing Times (in ms) for Random Algebraic Sign Com- putation	177
27	Empirical Computing Times (in ms) for Random Algebraic Sign Com- putation	178
28	Empirical Computing Times (in ms) for Random Algebraic Sign Com- putation	178
29	Empirical Comparison of Algebraic Polynomial Root Bound Algorithms	187
30	Empirical Behavior of AFPRIN for Random Polynomials	204
31	Empirical Behavior of AFPRIDS for Random Polynomials	216
32	Empirical Behavior of AFPRICS for Random Polynomials	222
33	Empirical Behavior of AFPRICSIR for Random Polynomials	224

34	Empirical Behavior of BRIPRICS for Random Polynomials	232
35	IPRICS Statistics for Random Resultants	236
36	IPRICS Statistics for Random Discriminants	237
37	IPRIST Statistics for Random Resultants	237
38	IPRIST Statistics for Random Discriminants	237
39	IPRIDS Statistics for Random Resultants	238
40	IPRIDS Statistics for Random Discriminants	238
41	AFPRICS Statistics for Random Resultants	241
42	Perturbed Wilkinson Polynomials	245
43	Perturbed Random Polynomials	247
44	Computing Times for CADs	248

CHAPTER I

Introduction

This thesis investigates algorithms for polynomial real root isolation of polynomials with integer and real algebraic number coefficients. Improved algorithms are derived and implemented for both integral and algebraic polynomials. Some of the improvements include fast algorithms for polynomial permutation and transformation, improved algorithms for computing the sign of a real algebraic number, algorithms for computing a root bound of a polynomial with real algebraic number coefficients, the use of multiple extensions, and the use of interval arithmetic. The algorithms are carefully compared both theoretically and empirically. Improved computing time bounds are obtained, strong conjectures are provided for the average computing times, and insight is given into the performance of the various algorithms. This thesis contains the first careful comparison and implementation of these algorithms for polynomials with real algebraic number coefficients.

A real root isolation algorithm computes isolating intervals (intervals containing exactly one root) for all of the real roots of a polynomial. Real root isolation is an essential and time consuming part of G. E. Collins' cylindrical algebraic decomposition (CAD) based quantifier elimination (QE) algorithm [18]. The improved algorithms presented in this thesis significantly reduce the computing time of Collins' algorithm.

CAD requires real root isolation of integral polynomials in the base case and of polynomials with real algebraic number coefficients in the extension phase. Real root isolation can be used to compute the number of real roots of a polynomial. Since this may not be defined for polynomials with approximate coefficients and since numerical algorithms might produce an error, we can not rely on a numerical algorithms. Therefore, it is essential that we have a real root isolation algorithm, which is exact, yet is as efficient as possible.

Polynomial real root isolation has a long and illustrious history. For a classical introduction to the subject, the reader should consult a book on the Theory of Equations such as the one by Burnside and Panton [8], Dickson [21], or Uspensky [48]. These books contain theorems for finding the roots of a polynomial that were meant to be used in hand computation. The algorithms that we will investigate are based on three classical theorems from the Theory of Equations.

The first algorithm uses Sturm sequences and is based on a theorem due to Sturm. Sturm's theorem can be used to compute the number of real roots that a polynomial has in a left-open and right-closed interval. Since a Sturm sequence for a polynomial can be computed this clearly leads to an algorithm for isolating the real roots of a polynomial. The idea is to start with an initial interval containing all of the real roots (this can be obtained from a root bound for the polynomial). Sturm's theorem is applied and if there are no roots in the interval no isolating intervals are returned. If there is one root, the initial interval is an isolating interval. If there is more than one root, the algorithm is recursively applied to the left and right subintervals. This

approach was first implemented and analyzed in 1970 by Heindel [24].

The second algorithm is based on the derivative sequence and Rolle's theorem. The basic idea behind the algorithm is to recursively compute isolating intervals for the roots of a polynomial from isolating intervals for the roots of its derivative. From Rolle's theorem and the intermediate value theorem, there is a root of a polynomial between two consecutive roots of its derivative if and only if the polynomial has opposite signs at the roots of the derivative. A benefit of this algorithm is that it provides isolating intervals for the roots of the derivatives. An algorithm based on this idea was carefully described and analyzed in a paper of Collins and Loos [14]. In [51], Zassenhaus uses a construction very similar to this to constructively prove the existence of a real closure of an algebraically ordered field. Zassenhaus attributes this construction to an earlier thesis of Hollkott [25].

This algorithm is related to another classical theorem independently due to Budan and Fourier, which states that the difference in the number of variations in the derivative sequence evaluated at the endpoints of an interval, gives an upper bound on the number of roots in the interval. The Collins-Loos algorithm suggests an alternative proof of this classical result. Furthermore, this theorem can be used to obtain an alternative derivation and an improved version of the algorithm.

The third algorithm, called the coefficient sign variation method, is based on Descartes' rule of signs (Descartes' stated the rule in *The Geometry*) and a sequence of polynomial transformations. Descartes' rule states that the number of coefficient sign variations of a polynomial is greater than or equal to the number of positive roots

of the polynomial and differs by an even number. Descarte's rule can be applied to obtain an algorithm similar to the one based on Sturm sequences. The algorithm used in this thesis can be traced to Uspensky's book and was first discussed in a paper of Collins and Akritas [2].

As in the Sturm sequence algorithm, this algorithm is based on repeatedly bisecting an initial interval until all of the roots have been isolated. For convenience, the positive and negative roots are isolated separately. If the number of coefficient sign variations is zero then the polynomial does not have any positive roots and if the number of variations is one then the interval from zero to a positive root bound is an isolating interval. If there is more than one variation then the interval containing the positive roots is bisected. A polynomial whose positive roots correspond to the roots in the left subinterval and a polynomial whose positive roots correspond to the roots in the right subinterval are computed. The algorithm is recursively applied to these two transformed polynomials.

It is not clear that this algorithm will terminate. However, it is possible, using a theorem due to Vincent [49], to show that the algorithm will eventually terminate. That is, an interval containing a single root will eventually give rise to a transformed polynomial with a single sign variation. An alternative proof of termination, which more accurately reflects the behavior of the algorithm, is presented in this thesis.

In this thesis we will review all three of these algorithms. Alternative derivations are used in the presentation of the algorithms. For each algorithm we discuss several variations, improvements, and efficient implementations. The presentation and im-

plementations are intended to facilitate careful theoretical and empirical comparisons. A major portion of this thesis is devoted to comparing the root isolation algorithms discussed above. This thesis contains careful theoretical and empirical comparisons of all of the algorithms for polynomials with both integer and real algebraic number coefficients.

Theoretically, it is possible to obtain maximum computing time bounds for the various algorithms. Using a generalization of a theorem due to Davenport [20] we obtain improved computing time bounds for all three algorithms. Davenport obtained the same improvement, in a slightly different way, for the Sturm sequence algorithm. He suggested that the technique would carry over to other algorithms. The best maximum computing time bound we were able to obtain is for the coefficient sign variation method. If m is the degree of the input polynomial and d is the sum of the absolute values of the coefficients, then $m^5L(d)^2$ is a computing time bound for the coefficient sign variation method when applied to integral polynomials. Using a class of polynomials due to Mignotte [38], we show that this bound is essentially the best that can be obtained for this algorithm.

If only the number of real roots is desired, it is possible to obtain a better bound. Using a modular version for Sturm sequence computation we obtain a bound of $m^3L(d)^2$ for calculating the number of real roots.

In practice the coefficient sign variation method typically is overwhelmingly faster than the other algorithms (several orders of magnitude). This is certainly the case for polynomials with random coefficients or polynomials with uniformly distributed

rational roots. Since most applications do not use random polynomials, it is necessary to compare the algorithms for other types of polynomials. Since our main application is CAD, we compared the algorithms with polynomials that arise in CAD, such as resultants, discriminants, and norms of algebraic polynomials. For almost all of the inputs we examined, the coefficient sign variation method performed significantly better than the other algorithms.

The Sturm sequence algorithm is usually considerably slower due to the large coefficients in the Sturm sequence and the high cost of computing the Sturm sequence. Even if the Sturm sequence has small coefficients, as is the case for orthogonal polynomials such as Chebycheff polynomials, the cost of evaluating the Sturm sequence still appears to be larger than the cost of the coefficient sign variation method. Nonetheless, it is possible to construct examples where the Sturm sequence algorithm is superior. Since the Sturm sequence algorithm is not affected by complex roots as is the coefficient sign variation method, or by roots of the derivative as is the derivative sequence method, the Sturm sequence algorithm can perform better for polynomials with a pair of complex conjugate roots that are close to x -axis or for polynomials which are nearly tangent to the x -axis at a root of the derivative. Using a class of polynomials related to Mignotte's we exhibit polynomials that have this behavior.

Even though we did our best to implement all of the algorithms as efficiently as possible, it is possible that a particular implementation could be improved, thereby leading to improved computing times and thus requiring a new comparison. Therefore, we attempted to use more information than just computing times in comparing

the algorithms. The amount of work the algorithms require can be characterized by information such as the number of bisections, the distribution of the roots, and the number of roots of the derivatives. For the Sturm sequence algorithm and the coefficient sign variation method, this information is nicely described using a binary tree corresponding to the search the algorithms perform. The height is related to the minimum distance between any two roots and the number of internal nodes is equal to the number of bisections. The number of nodes gives information on the number of Sturm sequence evaluations for the Sturm sequence algorithm and the number of polynomial transformations for the coefficient sign variation method. We recorded information on these trees such as their height and number of nodes for a large number of polynomials of varying degree and coefficient size. Using this information and a theorem due to Kac [27] on the average number of real roots of a polynomial, we are able to conjecture average computing times for the various algorithms.

The algorithms discussed above carry over to polynomials with real algebraic number coefficients provided we have an algorithm for computing the sign of a real algebraic number and an algorithm for computing a root bound for polynomials with real algebraic number coefficients. Alternatively it is possible, using the norm, to reduce the problem of isolating the real roots of a polynomial with real algebraic number coefficients to isolating the real roots of an integral polynomial. Since the roots of the norm contain the roots of the polynomial, an isolating interval for the norm is an isolating interval for the polynomial if and only if the polynomial has opposite signs at the endpoints of the interval.

A real algebraic number, α , is represented with a defining polynomial, $A(x)$, and an isolating interval $I = (a, b)$ containing α . The elements of the extension field of the rationals obtained by adjoining α are represented by polynomials. The sign of an element $\beta = B(\alpha) \in Q(\alpha)$ can be computed by refining the isolating interval for α until it does not contain any roots of $B(x)$. If this is the case, the sign of $B(x)$ is invariant for all $x \in I$, hence the sign of β is equal to the sign of $B(c)$, where c is the midpoint of I . This approach to sign computation has been previously considered by Rubald [42] and Rump [44]. We present several approaches for choosing the subinterval of I which contains α , and for determining if a polynomial has a root in an interval. The version with the best computing time can compute the sign of $B(\alpha)$ in time bounded by $m^5 L(d)^2$. This same computing time bound holds for the time to compute the signs of $B(\alpha)$ for all of the real conjugates of α .

Using this sign computation algorithm we show that the coefficient sign variation method can be used to isolate the real roots of a polynomial with coefficients in $Q(\alpha)$ in time bounded by $m^{11} n^6 L(d)^3$, where m is the degree of α , n is the degree of the polynomial, and d is a bound on the coefficients. The norm based algorithm, using the coefficient sign variation method for isolating the real roots of the norm, has computing time bounded by $m^{10} n^5 L(d)^2$. The bounds for these two algorithms based on the coefficient sign variation method are better than the bounds obtained for the other two algorithms. Furthermore, the coefficient sign variation method is significantly faster in practice, for the polynomials we examined, than the other algorithms, including the norm based algorithm.

These algorithms can be improved if each time a sign computation is performed, the refined isolating interval for the algebraic number is retained for future sign computations. This idea leads to an improved computing time bound for the coefficient sign variation method of $m^5n^5L(d)^2 + m^9n^4L(d)^2$. This idea also leads to a practical improvement in the computing time.

A further improvement can be obtained in many cases if interval arithmetic is used instead of exact arithmetic with algebraic numbers. The idea is to represent an algebraic number by an interval containing it. A polynomial with interval coefficients can be computed which contains a given polynomial with real algebraic number coefficients. After the interval polynomial is computed, the coefficient sign variation method can be used provided the resulting interval coefficients of the transformed polynomials do not contain zero. If an interval does not contain zero, its sign is well defined and is equal to the sign of the endpoints. For many polynomials, no undetermined coefficients will be obtained provided the width of the interval coefficients are small enough. However, if a coefficient equal to zero is encountered in one of the transformed polynomials, the algorithm will never succeed, no matter how small the width of the interval coefficients are. In our experiments, with 650 random polynomials, the interval arithmetic version of the coefficient sign variation method did not encounter any undetermined sign when using intervals coefficients with width equal to 2^{-10} . Only two failures occurred with widths equal to 2^{-5} , and only 27 failures were encountered with intervals as wide as one.

Whether or not the algorithm succeeds for a given interval polynomial is related

to the concept of an ill-conditioned polynomial (i.e. a polynomial such that a small change in the coefficients leads to a large change in the roots). In terms of root isolation, an ill-conditioned polynomial is a polynomial such that a small change in the coefficients leads to a different number of real roots. Using CAD we can determine the smallest change in the coefficients required to change the number of real roots. In this way we study the classic example of an ill-conditioned polynomial given by Wilkinson [50] and give examples that suggest that random polynomials are not ill-conditioned.

The algorithms in this thesis are described using an informal mathematical notation similar to the Aldes programming language [33] and the algorithm description format used by Knuth in his series on the Art of Computer Programming [29]. All of the algorithms described in this thesis have been implemented, in the C version of Aldes and consequently in C, using the SAC-2 Computer Algebra library [16]. Since we are extremely concerned with the efficiency of the computer implementation of the algorithms, fairly low level algorithms have been described in detail. Since many of the algorithms will be used by other algorithms as subalgorithms, the reader should consult the list of figures to look up various subalgorithm names.

Careful timings of the implementations of most of the algorithms described in this thesis have been included. All timings were obtained on a SPARCstation 1+, with 64 Megabytes of memory and rated at 15.8 mips. The timings were obtained with the UNIX timing procedure “times” and are reported in milliseconds. The times for many important subalgorithms are provided to help estimate and compare the computing

times of higher level algorithms.

The material in this thesis is organized into seven chapters. Chapter II introduces the notation we will use for computing time analyses and reviews the theorems that are needed in the derivation and analysis of root isolation algorithms. Chapter III discusses important auxiliary algorithms for interval operations, polynomial evaluation and transformation, and root bound computation. Chapter IV describes and analyzes root isolation algorithms for polynomials with integer coefficients. The chapter concludes with an extensive empirical comparison of the various algorithms. Chapter V contains algebraic number algorithms needed for real root isolation such as sign computation, polynomial transformations and evaluation, root bound computation, norm computation, and gcd and subresultant computation. Chapter VI contains a description and analysis of algorithms for root isolation of polynomials with real algebraic number coefficients. The chapter begins with a norm based algorithm which reduces the problem to root isolation of integral polynomials. Next, the same three algorithms that were used for integral polynomials, are carried over to polynomials with real algebraic number coefficients. The various algorithms are analyzed and compared, and several important improvements relevant to polynomials with real algebraic number coefficients are discussed and evaluated. An important improvement is based on the use of interval arithmetic. Chapter VII studies the use of root isolation in CAD and uses CAD to study ill-conditioned polynomials.

CHAPTER II

Mathematical Preliminaries

In this chapter we present the mathematical results that are needed to derive and analyze the real root isolation algorithms described in this thesis. Most of these results are old; however, they are listed for convenient reference. Proofs are given only when they are new or their presentation will help explain or compare the algorithms that appear in this thesis.

Section 2.1 introduces the notation that is used for computing time analyses. Section 2.2 discusses polynomial inequalities that are needed for the computing time analyses of real root isolation algorithms. In particular, bounds are given for the largest root of a polynomial and the minimum distance between any two roots. Section 2.3 introduces resultants, subresultants and polynomial remainder sequences (PRSs). Resultants can be used to perform certain algebraic number operations that are used to derive and analyze real root isolation algorithms for polynomials with real algebraic number coefficients. The fundamental theorem of PRSs is stated and used to obtain a bound on the size of the coefficients of the polynomials of a PRS. This information is needed to describe and analyze a root isolation algorithm based on Sturm sequences, which are a special type of PRS. Section 2.4 discusses linear fractional transformations and their effect when applied to the roots of a polynomial. These transformations

are an essential part of the coefficient sign variation method for real root isolation. Finally, Section 2.5 reviews some classic theorems used for counting the number of real roots in an interval. These theorems serve as the basis of the different root isolation algorithms.

2.1 Notation for Computing Time Analyses

In this section we review the concepts and notation that will be used for computing time analyses. We will be interested in the number of operations required by an algorithm as a function of its inputs. The inputs will be characterized by a set of parameters and the computing time will be given as a function of these parameters. For example, a polynomial is typically characterized by its degree and the size of its coefficients. The size of an integer is defined by its β -length.

Definition 1 (Length) *The β -length of a non-zero integer a , $L_\beta(a)$, is defined as the number of digits in its base β representation. The β -length of 0 is defined to be 1. Since the lengths of an integer using any two bases are codominant, the subscript β can be ignored in computing time analyses.*

The number of operations is given in terms of a dominance relation.

Definition 2 (Dominance and Codominance) *If f and g are two real-valued functions defined on a set S , f is dominated by g , $f \preceq g$, if there is a positive real number c such that $f(x) \leq cg(x)$ for all $x \in S$. If $f \preceq g$ and $g \preceq f$, f and g are said to be codominant, $f \sim g$. If $f \preceq g$ but not $g \preceq f$, f is strictly dominated by g , $f \prec g$.*

In discussing the computing time of an algorithm, we will be interested in the maximum and average computing time. The maximum computing time is the maximum number of operations for any of the finitely many valid inputs characterized by the specified parameters. The average computing time is the average number of operations over the same class of inputs.

Throughout this thesis we will use the computing times for the basic arithmetic and polynomial operations presented in [11, 17]. More complicated algorithms such as real root isolation depend on subalgorithms for polynomial and integer arithmetic, and their analysis will depend on the cost of these subalgorithms. Since we are interested in the implementation and analysis of practical algorithms for moderate size inputs, fast algorithms for integer multiplication and greatest common divisors will not be used.

2.2 Polynomial Inequalities

In this section we present some polynomial inequalities that are needed in the computing time analysis of real root isolation algorithms. Two basic inequalities are required. The first gives a bound on the largest root of a polynomial in terms of its degree and coefficients. The second gives a bound on the minimum distance between any two roots, called the minimum root separation. The minimum root separation can be used to bound the number of iterations a root isolation algorithm needs to separate all of the roots. We also give a bound, due to J. Davenport [20], on the product of the distances between certain roots. This bound can be used instead of the minimum root separation and it leads to improved computing time bounds.

We begin with some norms that are used to measure the size of the coefficients of a polynomial.

Definition 3 (Polynomial Seminorms)

Let $A(x) = A(x_1, \dots, x_r, x) = \sum_{i=0}^m a_i(x_1, \dots, x_r)x^i$ be a polynomial whose coefficients are complex polynomials in $r \geq 0$ variables. If $A(x)$ is a constant polynomial, its k norm, $|A(x)|_k$, is just the norm of a complex number. Otherwise, $|A(x)|_k$ is defined recursively: $|A(x)|_k = \{\sum_{i=0}^m |a_i|_k^k\}^{1/k}$. $|A(x)|_1$ is the sum norm, $|A(x)|_2$ is the Euclidean norm, and $|A(x)|_\infty = \max_{0 \leq i \leq m} |a_i|$ is the max norm.

Polynomial seminorms satisfy the following properties.

Theorem 1 (Norm Inequality) $(m+1)|A(x)|_\infty \geq |A(x)|_1 \geq |A(x)|_2 \geq \dots \geq |A(x)|_\infty$.

Theorem 2 (Norm Properties)

$$|A(x) + B(x)|_k \leq |A(x)|_k + |B(x)|_k$$

$$|A(x)B(x)|_\infty \leq |A(x)|_\infty |B(x)|_1$$

$$|A(x)B(x)|_1 \leq |A(x)|_1 |B(x)|_1.$$

The product of the roots of a polynomial, lying outside the unit circle, provides another useful polynomial measure.

Definition 4 (Polynomial Measure) Let $A(x) = a_m \prod_{i=1}^m (x - \alpha_i)$ be a polynomial with complex coefficients. The measure of $A(x)$, $M(A(x))$ is equal to

$$|a_m| \prod_{i=1}^m \max(1, |\alpha_i|).$$

Landau's theorem relates this measure to polynomial seminorms.

Theorem 3 (Landau's Inequality) $M(A(x)) \leq |A(x)|_2$.

PROOF. [39] ■

Landau's inequality can be used obtain a bound for the sizes of the coefficients of a factor of a polynomial.

Theorem 4 (Mignotte's Factor Coefficient Bound) Let $A(x) = \sum_{i=0}^m a_i x^i$ be a complex polynomial and let $B(x) = \sum_{j=0}^n b_j x^j$ be a divisor of $A(x)$. Then $|B(x)|_1 \leq 2^n |b_n/a_m| |A(x)|_2$.

PROOF. [39] ■

The Vandermonde determinant of the roots of a polynomial is very useful for deriving root separation theorems.

Definition 5 (Vandermonde Matrix) The Vandermonde matrix $V(\alpha_1, \dots, \alpha_m)$ is defined as

$$\begin{pmatrix} \alpha_1^{m-1} & \dots & \alpha_1 & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \alpha_m^{m-1} & \dots & \alpha_m & 1 \end{pmatrix}.$$

Theorem 5 (Vandermonde Determinant) $\det(V(\alpha_1, \dots, \alpha_m)) = \prod_{i < j} (\alpha_i - \alpha_j)$.

PROOF. [48] page 214 ■

The Vandermonde determinant is related to the discriminant of a polynomial.

Definition 6 (Discriminant) The discriminant of $A(x)$, $\text{disc}(A(x))$, is defined to be $a_m^{2m-2} \prod_{i < j} (\alpha_i - \alpha_j)$.

The discriminant can be expressed in terms of the coefficients of $A(x)$.

An estimate for the determinant of a matrix is given by Hadamard's inequality.

Theorem 6 (Hadamard's Inequality) *Let*

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,m} \end{pmatrix}$$

be a matrix with complex entries. Then

$$|\det(A)|^2 \leq \prod_{i=1}^m \left(\sum_{j=1}^m |a_{i,j}|^2 \right).$$

PROOF. [40] page 114 ■

Hadamard's inequality can be extended to matrices with polynomial entries.

Theorem 7 (Polynomial Analog of Hadamard's Theorem) *Let M an $m \times m$ matrix whose entries are polynomials in r variables with complex coefficients. Then*

1. $|\det(A)|_1 \leq \prod_{i=1}^m |M_i|_1,$
2. $|\det(A)|_2 \leq \prod_{i=1}^m |M_i|_2,$

where M_i is the i th row of M .

PROOF. Part 1 is proven in [12] and Part 2 can be proven using a generalization of the proof of a similar result in [22] ■

2.2.1 Root bounds

In this section we review several theorems that give bounds on the roots of a polynomial. These bounds are used in the design and analysis of real root isolation algorithms.

Definition 7 (Root Bound) Let $A(x)$ be a polynomial with complex coefficients. A root bound for the polynomial $A(x)$ is a real number B such that if $A(\alpha) = 0$ then $|\alpha| < B$.

There are many root bound theorems, of which we will use the following two.

Theorem 8 (Cauchy) Let $A(x) = \sum_{i=0}^m a_i x^i$ be a polynomial with complex coefficients.

$$B = 1 + \frac{|A(x)|_\infty}{|a_m|}$$

is a root bound for $A(x)$.

PROOF. Assume that $A(\alpha) = 0$. If $|\alpha| \leq 1$ the theorem is true, so we can assume that $|\alpha| > 1$. Then

$$\begin{aligned} |\alpha|^m &\leq (1/|a_m|) \sum_{i=0}^{m-1} |a_i| |\alpha|^i \\ &\leq (|A(x)|_\infty/|a_m|) \sum_{i=0}^{m-1} |\alpha|^i \\ &\leq (|A(x)|_\infty/|a_m|) \frac{|\alpha|^m - 1}{|\alpha| - 1} \\ &< (|A(x)|_\infty/|a_m|) \frac{|\alpha|^m}{|\alpha| - 1}. \end{aligned}$$

Multiplying both sides of this inequality by $(|\alpha| - 1)/|\alpha|^m$ and adding 1 to each side shows that $|\alpha| < 1 + |A(x)|_\infty/|a_m|$ ■

Theorem 9 (Knuth) Let $A(x) = \sum_{i=0}^m a_i x^i$ be a polynomial with complex coefficients. If

$$B = \max_{1 \leq k \leq m} |a_{m-k}/a_m|^{1/k},$$

then $2B$ is a root bound for $A(x)$.

PROOF. Assume $A(\alpha) = 0$ and $|\alpha| > B$. Then

$$|\alpha|^m \leq (1/|a_m|) \sum_{i=0}^{m-1} |a_i| |\alpha|^i, \quad (2.1)$$

which implies

$$1 \leq (1/|a_m|) \sum_{i=0}^{m-1} |a_i| |\alpha|^{i-m}, \quad (2.2)$$

Setting $k = m - i$ we obtain

$$1 \leq (1/|a_m|) \sum_{k=1}^m \frac{|a_{m-k}|}{|\alpha|^k} \leq \sum_{k=1}^m \left(\frac{B}{|\alpha|} \right)^k. \quad (2.3)$$

This sum can be estimated with an infinite geometric sum, which implies that

$$1 < \frac{B}{|\alpha|} \left(\frac{|\alpha|}{|\alpha| - B} \right) = \frac{B}{|\alpha| - B},$$

which proves the theorem ■

The proof in Theorem 9 can be modified to obtain a bound for the positive roots of a real polynomial.

Theorem 10 (Positive Root Bound) *Let $A(x) = \sum_{i=0}^m a_i x^i$ be a polynomial with real coefficients. If*

$$B = \max_{a_{m-k}/a_m < 0} |a_{m-k}/a_m|^{1/k}$$

then $2B$ is a bound for the positive roots of $A(x)$.

2.2.2 Root Separation Theorems

In this section we derive a bound on the minimum distance between any two roots of a polynomial.

Definition 8 (Minimum Root Separation.) Let $A(x) = \sum_{i=0}^m a_i x^i$, $m \geq 2$, be a polynomial with complex coefficients. The minimum root separation of a polynomial $A(x)$, $\text{sep}(A(x))$, is defined as $\min_{\alpha_i \neq \alpha_j} |\alpha_i - \alpha_j|$.

The following theorem gives a bound on the product of the distances between certain roots of a polynomial. This is a generalization of a theorem due to Davenport [20], which in turn is a generalization of a theorem due to Mahler [36].

Theorem 11 (Davenport) Let $A(x) = \sum_{i=0}^m a_i x^i = a_m \prod_{i=1}^m (x - \alpha_i)$ be a polynomial with complex coefficients. Let $\beta_1, \dots, \beta_k$ be a subset of the roots of $A(x)$ with $|\beta_i| \leq |\alpha_i|$ and $\beta_i \notin \{\alpha_1, \dots, \alpha_i\}$. Let $D = |\text{disc}(A(x))|$ and $M = M(A(x))$ the measure of $A(x)$ (4). Then

$$\prod_{i=1}^k |\alpha_i - \beta_i| \geq 3^{k/2} D^{1/2} M^{-m+1} m^{-k-m/2}.$$

PROOF. We can assume $A(x)$ is squarefree since if $A(x)$ is not squarefree its discriminant is zero and the theorem is trivially true. Therefore $\alpha_1, \dots, \alpha_m$ are distinct. Let V be the Vandermonde matrix $V(\alpha_1, \dots, \alpha_m)$. By Definition 6 and Theorem 5, $D^{1/2} = |a_m|^{m-1} |\det(V)|$. For $i = 1, 2, \dots, k$, subtract row $j(i)$ from row i , where $\beta_i = \alpha_{j(i)}$, obtaining the matrix

$$V' = \begin{pmatrix} \alpha_1^{m-1} - \beta_1^{m-1} & \dots & \alpha_1 - \beta_1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ \alpha_k^{m-1} - \beta_k^{m-1} & \dots & \alpha_k - \beta_k & 0 \\ \alpha_{k+1}^{m-1} & \dots & \alpha_k & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \alpha_m^{m-1} & \dots & \alpha_m & 1 \end{pmatrix}.$$

Since $\alpha_i^{h+1} - \beta_i^{h+1} = (\alpha_i - \beta_i) \sum_{j=0}^h \alpha_i^{h-j} \beta_i^j$,

$$D^{1/2} / |a_m|^{m-1} = \left\{ \prod_{i=1}^k |\alpha_i - \beta_i| \right\} |\det(V'')|, \quad (2.4)$$

where

$$V'' = \begin{pmatrix} \sum_{j=0}^{m-2} \alpha_1^{m-2-j} \beta_1^j & \cdots & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ \sum_{j=0}^{m-2} \alpha_k^{m-2-j} \beta_k^j & \cdots & 1 & 0 \\ \alpha_{k+1}^{m-1} & \cdots & \alpha_k & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \alpha_m^{m-1} & \cdots & \alpha_m & 1 \end{pmatrix}.$$

The desired inequality is obtained by applying Hadamard's inequality (Theorem 6) to $|\det(V'')|$. To apply Hadamard's inequality we need a bound on the Euclidean norms of the rows of V'' . Since $|\beta_i| \leq |\alpha_i|$

$$\begin{aligned} \left| \sum_{j=0}^h \alpha_i^{h-j} \beta_i^j \right| &\leq \sum_{j=0}^h |\alpha_i^{h-j} \beta_i^j| \\ &\leq \sum_{j=0}^h |\alpha_i|^h = (h+1) |\alpha_i|^h, \end{aligned}$$

and for $1 \leq i \leq k$,

$$|V_i''|_2 \leq \max(1, |\alpha_i|)^{m-2} \left(\sum_{h=1}^{m-1} h^2 \right)^{1/2} \leq (m^3/3)^{1/2} \max(1, |\alpha_i|)^{m-1},$$

where V_i'' is the i -th row of V'' . For $i > k$, $|V_i''|_2 \leq m^{1/2} \max(1, |\alpha_i|)^{m-1}$.

Hadamard's inequality then implies

$$\begin{aligned} |\det(V'')| &\leq \prod_{i=1}^k (m^{3/2}/\sqrt{3}) \max(1, |\alpha_i|)^{m-1} \prod_{i=k+1}^m m^{1/2} \max(1, |\alpha_i|)^{m-1} \\ &= 3^{-k/2} m^{m/2+k} (M/|a_m|)^{m-1}. \end{aligned}$$

Inserting this into Equation 2.4 completes the proof ■

Corollary 1 (Integral Polynomials) *Assume the same hypotheses as Theorem 11.*

Furthermore assume that $A(x)$ has integral coefficients, and let $d = |A(x)|_2$. Then

$$\prod_{i=1}^k |\alpha_i - \beta_i| \geq 3^{k/2} d^{-m+1} m^{-k-m/2}.$$

PROOF. If $A(x)$ is a squarefree integral polynomial, its discriminant is a non-zero integer, so $D^{1/2} \geq 1$. The proof is completed since $M \leq |A(x)|_2$ by Landau's inequality (Theorem 3) ■

Corollary 2 (Mahler's Root Separation Theorem) *Let $A(x)$ be a squarefree polynomial with complex coefficients. Then*

$$\text{sep}(A(x)) \geq \sqrt{3}m^{-1-m/2}D^{1/2}d^{-m+1}.$$

If $A(x)$ is an integral polynomial, then

$$\text{sep}(A(x)) \geq \sqrt{3}m^{-1-m/2}d^{-m+1}.$$

PROOF. Apply Theorem 11 and Corollary 1 with $k = 1$ and α_1 and β_1 , with $|\alpha_1| \leq |\beta_1|$, the two closest roots ■

Corollary 3 (Product of Real Roots) *Let $\gamma_1, \dots, \gamma_{k+1}$ be any of the real roots of $A(x)$ with $\gamma_1 < \gamma_2 < \dots < \gamma_{k+1}$. Then*

$$\prod_{i=1}^k |\gamma_i - \gamma_{i+1}| \geq 3^{k/2} D^{1/2} d^{-m+1} m^{-k-m/2}.$$

PROOF. Assume $\gamma_1 < \dots < \gamma_j < 0 < \gamma_{j+1} < \dots < \gamma_k$. We can further assume that $|\gamma_{j+1}| \geq |\gamma_j|$ since we can multiply all of the roots by -1 if this assumption is not true. Let $\alpha_i = \gamma_i$ and $\beta_i = \gamma_{i+1}$ for $1 \leq i < j$. Let $\alpha_i = \gamma_{k+j-i}$ and $\beta_i = \gamma_{k+j-i-1}$ for $j \leq i < k$ and apply Theorem 11. The theorem is proved since $\prod_{i=1}^k |\alpha_i - \beta_i| = \prod_{i=1}^k |\gamma_i - \gamma_{i+1}|$ ■

Corollary 4 (Product of Complex Conjugates) *Let $\gamma_1, \gamma_2, \dots, \gamma_{2k-1}, \gamma_{2k}$ be k pairs of complex conjugate roots, with $\gamma_{2i} = \overline{\gamma_{2i-1}}$, the complex conjugate of γ_{2i-1} . Then*

$$\prod_{i=1}^k |\gamma_{2i-1} - \gamma_{2i}| \geq 3^{k/2} D^{1/2} d^{-m+1} m^{-k-m/2}.$$

PROOF. Let $\alpha_i = \gamma_{2i-1}$ and $\beta_i = \gamma_{2i}$ for $1 \leq i \leq k$ and apply Theorem 11 ■

2.3 Resultants, Subresultants and Polynomial Remainder Sequences

The resultant of two polynomials can be used to perform many important operations with algebraic numbers [34]. Some of these operations are needed in the analysis and derivation of real root isolation algorithms for polynomials with real algebraic number coefficients. In this section we review the properties that we will need. We also review the definition of a generalization of the resultant called a subresultant. Subresultants can be used to bound the sizes of the coefficients of the polynomials in a polynomial remainder sequence (PRS).

A PRS for two polynomials $A(x)$ and $B(x)$ is a generalization of the remainder sequence used to compute the greatest common divisor of $A(x)$ and $B(x)$ using the Euclidean algorithm. A general PRS is introduced to avoid computation in the field of fractions. For example, it is desirable to compute a PRS of two integral polynomials without computing with rational numbers since rational number operations require the computation of many integer gcds, which can be costly. PRSs are useful for real root isolation algorithms since a Sturm sequence is a special type of PRS. In this section we review the important definitions and properties of PRSs. More detailed

information on PRSs can be found in the paper by Loos [35] and the paper by Brown and Traub [7].

Definition 9 (Resultant) Let $A(x) = \sum_{i=0}^m a_i x^i$ and $B(x) = \sum_{j=0}^n b_j x^j$ be polynomials with coefficients in a commutative ring. The resultant of $A(x)$ and $B(x)$, $\text{res}(A(x), B(x))$ is the determinant of the $(m+n) \times (m+n)$ Sylvester matrix.

$$\text{res}(A(x), B(x)) = \begin{vmatrix} a_m & \cdots & a_1 & a_0 & & \\ & \ddots & & \ddots & \ddots & \\ & & a_m & \cdots & a_1 & a_0 \\ b_n & \cdots & b_1 & b_0 & & \\ & \ddots & & \ddots & \ddots & \\ & & b_n & \cdots & b_1 & b_0 \end{vmatrix}$$

Theorem 12 Let $A(x) = \sum_{i=0}^m a_i x^i$ and $B(x) = \sum_{j=0}^n b_j x^j$ be polynomials with coefficients in a commutative ring. Then there exists polynomials $U(x)$ and $V(x)$ with $\deg(U(x)) < n$ and $\deg(V(x)) < m$ such that $\text{res}(A(x), B(x)) = A(x)U(x) + B(x)V(x)$.

PROOF. [34] ■

Theorem 13 Let $A(x) = a_m \prod_{i=1}^m (x - \alpha_i)$ where α_i is an indeterminant and $B(x) = \sum_{j=0}^n b_j x^j$, be polynomials with coefficients in an integral domain. Then $\text{res}(A(x), B(x)) = a_m^n \prod_{i=1}^m B(\alpha_i)$.

PROOF. Let $R = \text{res}(A(x), B(x))$. By Theorem 12, $B(\alpha_i) | R$ for $1 \leq i \leq m$. Viewing $B(x)$ as a polynomial with coefficients in the field of fractions of the integral domain, $\gcd(B(\alpha_i), B(\alpha_j)) = 1$ for $i \neq j$. Therefore, $\{\prod_{i=1}^m B(\alpha_i)\} | R$. Since $\deg_{\alpha_i}(R) = \deg_{\alpha_i}(\prod_{i=1}^m B(\alpha_i))$, $R = c \prod_{i=1}^m B(\alpha_i)$, where c is a constant. Setting $\alpha_1 = \cdots = \alpha_m = 0$, shows that $c = a_m^n$ ■

Theorem 14 (Norm Computation)

Let $A(x) = \sum_{i=0}^m a_i x^i$ and $B(x, y) = \sum_{j=0}^n b_j(x) y^j$, with $b_n(\alpha) \neq 0$. Then

$$R(y) = \text{res}_x(A(x), B(x, y)) = a_m^n \prod_{i=1}^m B(\alpha_i, y)$$

and $\deg(R(y)) = mn$.

PROOF. The proof follows from Theorem 13 ■

Theorem 15 (Minimal Polynomial Computation) Let $A(x) = \sum_{i=0}^m a_i x^i$ and $B(y) = \sum_{j=0}^n b_j y^j$. Then $R(y) = \text{res}_x(A(x), x - B(y)) = a_m^n \prod_{i=1}^m (y - B(\alpha_i))$.

PROOF. This is a special case of Theorem 14 ■

The subresultant is a polynomial whose coefficients are determinants of submatrices of the Sylvester matrix.

Definition 10 (Subresultant) Let $m = \deg(A(x))$, $n = \deg(B(x))$, and $0 \leq k < \min(m, n)$. Let M_k be the $(m + n - 2k) \times (m + n)$ matrix obtained by deleting the last k rows containing coefficients of $A(x)$ and the last k rows containing coefficients of $B(x)$. Let $M_{k,j}$ be the $(m + n - 2k) \times (m + n - 2k)$ matrix consisting of the first $m + n - 2k - 1$ columns of M_k followed by the $(m + n - 2k - j)$ -th column of M_k . Then $S_k(A(x), B(x)) = \sum_{j=0}^k \det(M_{k,j}) x^j$ is the k -th subresultant of $A(x)$ and $B(x)$.

The coefficients of the k -th subresultant can be bounded using the generalization of Hadamard's inequality.

Theorem 16 (Subresultant Coefficient Bound) Let $\deg(A(x)) = m$ and $\deg(B(x)) = n$. Then $|S_k(A(x), B(x))|_\infty \leq |A(x)|_1^{n-k} |B(x)|_1^{m-k}$.

PROOF. Apply Theorem 7 to the definition of $S_k(A(x), B(x))$ ■

The polynomials of any PRS for $A(x)$ and $B(x)$ are similar to subresultants of $A(x)$ and $B(x)$.

Definition 11 *Polynomials $A(x)$ and $B(x)$ are similar, $A(x) \sim B(x)$, if there exist non-zero a and b such that $aA(x) = bB(x)$.*

Definition 12 (Polynomial Remainder Sequence (PRS)) *A Polynomial Remainder Sequence for $A_1(x)$ and $A_2(x)$ is a sequence of polynomials*

$\{A_1(x), A_2(x), \dots, A_r(x), A_{r+1}(x) = 0\}$ with $e_i A_i(x) = Q_i(x) A_{i+1}(x) + f_i A_{i+2}(x)$, $\deg(A_{i+2}(x)) < \deg(A_{i+1}(x))$, and e_i and f_i non-zero.

Definition 13 (Degree Sequence and Normal PRS)

Let $S = \{A_1(x), A_2(x), \dots, A_r(x), A_{r+1}(x) = 0\}$ be a PRS, and let $m_i = \deg(A_i(x))$ and $\delta_i = m_i - m_{i+1}$. Then $\{m_1, \dots, m_r\}$ is the degree sequence of S and S is normal if $\delta_i = 1$ for $i = 1, \dots, r - 1$.

A special type of PRS is obtained by using the pseudo-remainder. The pseudo-remainder is a generalization of the remainder which does not need division.

Definition 14 (Pseudo-remainder) *Let $A(x)$ and $B(x)$ be polynomials with coefficients in a commutative ring with $\deg(A(x)) \geq \deg(B(x))$. The pseudo-remainder of $A(x)$ and $B(x)$, $\text{prem}(A(x), B(x))$, is the polynomial whose degree is less than $\deg(A(x))$ and satisfies $b_n^{m-n+1} A(x) = Q(x) B(x) + \text{prem}(A(x), B(x))$.*

For a given pair of integral polynomials, the size of the coefficients of the primitive PRS are the smallest possible for any PRS.

Definition 15 (Content and Primitive part) Assume $A(x)$ is a polynomial whose coefficients belong to a unique factorization domain. The content of $A(x)$, $\text{cont}(A(x))$, is the greatest common divisor of the coefficients of $A(x)$. The primitive part of $A(x)$, $\text{pp}(A(x))$, is equal to $A(x)/\text{cont}(A(x))$.

Definition 16 (Primitive PRS) A primitive PRS is a PRS $\{A_1(x), A_2(x), \dots, A_r(x), A_{r+1}(x) = 0\}$, where $e_i = \text{ldcf}(A_i(x))^{\delta_i+1}$, $f_i = \pm \text{cont}(A_{i+2}(x))$, and $A_i(x) = \pm \text{pp}(\text{prem}(A_i(x), A_{i+1}(x)))$.

The similarity between the polynomials in a PRS for $A_1(x)$ and $A_2(x)$ and subresultants of $A_1(x)$ and $A_2(x)$ is given in the following theorem.

Theorem 17 (Fundamental Theorem of PRSs)

Let $\{A_1(x), A_2(x), \dots, A_r(x), A_{r+1}(x) = 0\}$ be a PRS defined by $e_i A_i(x) = Q_i(x) A_{i+1}(x) + f_i A_{i+2}(x)$. Let $c_i = \text{ldcf}(A_i(x))$, $m_i = \deg(A_i(x))$, and $\delta_i = m_i - m_{i+1}$. Then

$$\begin{aligned} & \left\{ \prod_{i=2}^{k-1} (-1)^{(m_{i-1}-m_k)(m_i-m_k)} f_{i-1}^{m_i-m_k} c_i^{\delta_{i-1}-1} \right\} c_k^{\delta_{k-1}-1} A_k(x) \\ &= \left\{ \prod_{i=2}^{k-1} e_{i-1}^{m_i-m_k} \right\} S_{m_k} \quad (3 \leq k \leq r) \end{aligned} \quad (2.5)$$

$$\begin{aligned} & \left\{ \prod_{i=2}^{k-1} (-1)^{(m_{i-1}-m_{k-1}+1)(m_i-m_{k-1}+1)} f_{i-1}^{m_i-m_{k-1}+1} c_i^{\delta_{i-1}+\delta_i-1} \right\} A_k(x) \\ &= \left\{ \prod_{i=2}^{k-1} e_{i-1}^{m_i-m_{k-1}+1} \right\} c_{k-1}^{\delta_{k-1}-1} S_{m_{k-1}-1} \quad (3 \leq k \leq r) \end{aligned} \quad (2.6)$$

$$S_j = 0 \text{ for } m_k < j < m_{k-1} - 1 \quad (3 \leq k \leq r) \quad (2.7)$$

$$S_j = 0 \text{ for } 0 \leq j \leq m_r. \quad (2.8)$$

PROOF. [7] or [35] ■

This theorem can be used to bound the size of the coefficients in a PRS, using the bound on the coefficients of a subresultant. Moreover, there exists a PRS whose elements are subresultants.

Definition 17 (Subresultant PRS) *The Subresultant PRS is the PRS*

$\{G_1(x), G_2(x), \dots, G_r(x), G_{r+1}(x) = 0\}$ defined by $e_i = g_{i+1}^{\delta_i+1}$ and $f_1 = 1$ and $f_i = -g_i(-h_i)^{\delta_i}$ for $i > 1$, where $g_i = \text{lcf}(G_i(x))$ and $h_1 = 1$ and $h_i = g_i^{\delta_{i-1}} h_{i-1}^{\delta_{i-1}+1}$ for $i > 1$.

It can be shown (see [35]) that $G_i(x) = S_{n_{i-1}-1}(G_1(x), G_2(x))$ and that $G_{i+2}(x) = \text{prem}(G_i(x), G_{i+1}(x)) / (-g_i(-h_i)^{\delta_i})$. The subresultant PRS was first considered by Collins [9] and this form was first discovered by Brown [6]. The subresultant PRS is sometimes called the Subresultant PRS of the second kind to distinguish it from the Subresultant PRS of the first kind.

Definition 18 (Subresultant PRS of the First Kind) *The Subresultant PRS of the first kind is $\{G_1, G_2(x), S_{n_s}(G_1(x), G_2(x)), \dots, S_{n_r}(G_1(x), G_2(x))\}$.*

This PRS can be obtained from the subresultant PRS using the relationship (see [35]) $g_i S_{n_{i-1}-1}(G_1(x), G_2(x)) = h_i S_{n_i}(G_1(x), G_2(x))$.

2.4 Polynomial Transformations

In this section we introduce a class of transformations that will be used to obtain transformed polynomials whose roots in one region of the complex plane correspond to the roots of the original polynomial in another region of the complex plane. These

transformations are an essential part of the coefficient sign variation method (Section 4.2.1).

Definition 19 (Linear Fractional Transformation) *Let z be a complex number or ∞ . A linear fractional transformation is a mapping $z \longrightarrow (az+b)/(cz+d)$, where*

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} \neq 0.$$

This linear fractional transformation maps ∞ to a/c and $z = -d/c$ to ∞ .

The collection of linear fractional transformations form a group under composition.

Theorem 18 *The group of linear fractional transformations is isomorphic to the group of 2×2 non-singular matrices modulo the non-zero scalar matrices.*

PROOF. Let

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

and

$$N = \begin{pmatrix} e & f \\ g & h \end{pmatrix}.$$

Let L be the mapping from 2×2 matrices to linear fractional transformations defined by $L : M \longrightarrow L_M$, where $L_M : z \longrightarrow (az+b)/(cz+d)$.

Then

$$\begin{aligned} L_M(L_N z) &= L_M \left(\frac{ez+f}{gz+h} \right) \\ &= \frac{a \left(\frac{ez+f}{gz+h} \right) + b}{c \left(\frac{ez+f}{gz+h} \right) + d} \\ &= \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix} z \\ &= L_{MN}(z) \end{aligned}$$

A similar computation shows that $L_{MN}(z) = L_M(L_N(z))$ for $z = \infty$ or $z = -h/g$. Therefore L is a homomorphism mapping the 2×2 non-singular matrices onto the linear fractional transformations. Since the kernel of L is the non-zero scalar matrices, the proof is completed ■

Theorem 19 *The group of linear fractional transformations is generated by three special types of linear fractional transformations.*

1. *Translations.* $T_b : z \longrightarrow z + b$.
2. *Homothetic transformations.* $H_a : z \longrightarrow az$.
3. *Inversion.* $R : z \longrightarrow 1/z$.

PROOF.

By Theorem 18 computations with linear fractional transformations can be performed with matrices. We use the following matrix representations:

$$T_b = \begin{pmatrix} 1 & b \\ 0 & 1 \end{pmatrix}, H_a = \begin{pmatrix} a & 0 \\ 0 & 1 \end{pmatrix}, R = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

We begin with the special case of upper triangular matrices.

$$\begin{pmatrix} a & b \\ 0 & c \end{pmatrix} = \begin{pmatrix} 1 & b/c \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a & 0 \\ 0 & c \end{pmatrix}.$$

so that the linear fractional transformation corresponding to the upper triangular matrix above is equal to $T_{b/c}H_{a/c}$.

Using the inversion we get

$$\begin{pmatrix} d & e \\ f & 0 \end{pmatrix} = \begin{pmatrix} e & d \\ 0 & f \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

so that linear fractional transformations of this form are equal to $T_{d/f}H_{e/f}R$.

The proof is completed since every matrix is a product of an upper triangular matrix and a matrix of this form.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} a & 1 \\ c & 0 \end{pmatrix} \begin{pmatrix} 1 & d/c \\ 0 & b - ad/c \end{pmatrix} \blacksquare$$

The action of a linear fractional transformation on a polynomial is obtained by applying the linear fractional transformation to the roots of the polynomial. This transformation is performed by substituting the variable x with the inverse linear fractional transformation applied to x . The linear fractional transformation, L_M , is applied to a variable x by replacing x with $L_M(x) = (ax + b)/(cx + d)$. Unlike linear fractional transformations, linear fractional substitutions compose from left to right when viewed as matrices: $L_{MN}(x) = L_N(L_M(x))$. In general, a rational function is obtained after applying the substitution, $L_{M^{-1}}(x)$ in the polynomial $A(x)$. Therefore the denominator must be cleared to obtain a polynomial.

Definition 20 (Polynomial Transformation) Let $A(x) = \sum_{i=0}^m a_i x^i$ and

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$

Then $L_M(A(x)) = (-cx + d)^m A(L_{M^{-1}}(x))$.

It easily follows from the definition that $L_{MN}(A(x)) = L_M(L_N(A(x)))$. Let $\tilde{A}(x) = L_M(A(x))$. Then $\tilde{A}(L_M(\alpha)) = (-cL_M(\alpha) + d)^m A(L_{M^{-1}}(L_M(\alpha))) = (-cL_M(\alpha) + d)^m A(\alpha)$. Therefore, if $A(\alpha) = 0$ and $L_M(\alpha) \neq \infty$, then $\tilde{A}(L_M(\alpha)) = 0$.

Lemma 1 Let $A(x) = a_m \prod_{i=1}^m (x - \alpha_i)$, $\tilde{A}(x) = L_M(A(x))$, and

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

be non-singular. Assume $\prod_{i=1}^m (c\alpha_i + d) \neq 0$. If $c \neq 0$ then $\tilde{A}(x) = (-c)^m A(-d/c) \prod_{i=1}^m (x - L_M(\alpha_i))$. If $c = 0$ then $\tilde{A}(x) = d^m \prod_{i=1}^m (x - L_M(\alpha_i))$.

PROOF.

$$\begin{aligned} L_M(A(x)) &= (-cx + a)^m A(L_{M^{-1}}(x)) \\ &= (-cx + a)^m a_m \prod_{i=1}^m (L_{M^{-1}}(x) - \alpha_i) \\ &= (-cx + a)^m a_m \prod_{i=1}^m ((dx - b)/(-cx + a) - \alpha_i) \\ &= a_m \prod_{i=1}^m ((dx - b) - \alpha_i(-cx + a)) \\ &= a_m \prod_{i=1}^m ((c\alpha_i + d)x - (a\alpha_i + b)) \\ &= a_m \prod_{i=1}^m (c\alpha_i + d) \prod_{i=1}^m (x - (a\alpha_i + b)/(c\alpha_i + d)) \\ &= a_m \prod_{i=1}^m (c\alpha_i + d) \prod_{i=1}^m (x - L_M(\alpha_i)) \end{aligned}$$

If $c \neq 0$ then $\prod_{i=1}^m (c\alpha_i + d) = (-c)^m A(-d/c)$, and if $c = 0$ then $\prod_{i=1}^m (c\alpha_i + d) = d^m$ ■

Theorem 19 implies that a general polynomial transformation can be performed using polynomial translations, homothetic transformations, and inversions. Algorithms for implementing these special polynomial transformations are presented in Section 2.4. The following theorem gives a bound on the size of the coefficients of a transformed polynomial.

Theorem 20 *Let $A(x) = \sum_{i=0}^m a_i x^i$ be an integral polynomial and let a and b be integers.*

1. $|A(x+b)|_\infty \leq |A(x)|_\infty \max(1, |b|)(|b|+1)^m.$
2. $|A(x \pm 1)|_\infty \leq 2^m |A(x)|_\infty.$
3. $|A(ax)|_\infty \leq |a|^m |A(x)|_\infty.$
4. $|a^m A(x/a)|_\infty \leq |a|^m |A(x)|_\infty.$
5. $|R(A(x))|_\infty = |A(x)|_\infty.$

PROOF. Inequalities 3 and 4 are obvious. Since $R(A(x))$ is obtained by reversing the coefficients of $A(x)$ Inequality 5 is true.

Inequality 1 can be proven using Theorem 2.

$$\begin{aligned}
 |A(x+w)|_\infty &= \left| \sum_{i=0}^m a_i (x+b)^i \right|_\infty \\
 &\leq |A(x)|_\infty \sum_{i=0}^m |(x+b)^i|_\infty \\
 &\leq |A(x)|_\infty |x+b|_\infty \sum_{i=1}^m |(x+b)|_1^{i-1} \\
 &\leq |A(x)|_\infty |x+b|_\infty |x+b|_1^m \\
 &= |A(x)|_\infty \max(1, |b|)(1+|b|)^m
 \end{aligned}$$

Inequality 2 follows immediately from Inequality 1 ■

2.5 Root Counting Theorems

In this section we present some theorems for determining the number of real roots in an interval. Each of these theorems serves as the basis of a different real root isolation

algorithm. The first theorem, due to Sturm, gives the exact number of real roots in an interval and it is the basis of the first algorithm. Unfortunately the application of this theorem is costly, so alternative, less precise, theorems are sought.

The second theorem, due to Budan and Fourier, gives a upper bound for the number of real roots in an interval, and its application is less costly than Sturm's theorem. Although it may not give the exact answer, this deficiency can be remedied in two ways. The first uses an immediate corollary of the theorem of Budan and Fourier called Descartes' rule of signs. The key to its successful application is the use of polynomial transformations which move the roots apart. When the roots have been sufficiently separated Descartes' rule will yield the exact result. The second way of removing the shortcomings of the theorem of Budan and Fourier is to ensure that there are no roots of any of the derivatives in the interval in question. If this is the case, the application of the theorem of Budan and Fourier reduces to the intermediate value theorem and Rolle's theorem, and leads to an inductive algorithm which obtains isolating intervals for a polynomial from isolating intervals of its derivative.

Sturm's theorem relies on a sequence of polynomial called a Sturm sequence.

Definition 21 (Sturm Sequence) *A sequence of polynomials*

$A_1(x) = A(x), A_2(x), \dots, A_r(x)$ *is a Sturm sequence for the interval $[a, b]$ if the following properties hold.*

1. *If $\alpha \in [a, b]$ and $A(\alpha) = 0$, then there exists an $\epsilon > 0$ such that $A_1(x)A_2(x) < 0$ for $x \in (\alpha - \epsilon, \alpha)$ and $A_1(x)A_2(x) > 0$ for $x \in (\alpha, \alpha + \epsilon)$.*
2. *If $\alpha \in [a, b]$ and $A_k(\alpha) = 0$ for some $0 < k < r$, then $A_{k-1}(\alpha)A_{k+1}(\alpha) < 0$.*

3. For all $x \in [a, b]$, $A_r(x) \neq 0$.

From the definition, it is easy to see that a Sturm sequence for an interval is also a Sturm sequence for any subinterval. This is important for their application to real root isolation algorithms. The most important property of Sturm sequences is given by the following theorem.

Theorem 21 (Sturm's Theorem) *Let $A(x)$ be a polynomial with real coefficients. Let $A(x) = A_1(x), A_2(x), \dots, A_r(x)$ be a Sturm sequence for the interval $[a, b]$. Let V_x be the number of sign variations in the Sturm sequence evaluated at x . Then $V_a - V_b$ is equal to the number of distinct roots of $A(x)$ in the interval $(a, b]$.*

PROOF. Partition the interval $[a, b]$ by the real roots of the polynomials $A_i(x)$. By the intermediate value theorem, between any two roots the signs of $A_i(x)$ are invariant and hence V_x is invariant. Therefore we only need to consider the behavior of V_x in passing through a root of $A_i(x)$. There are two cases to consider. First assume that $\alpha \in [a, b]$ and $A_j(\alpha) = 0$ for some $1 < j < r$. By property 2 of a Sturm sequence $A_{j-1}(x)A_{j+1}(x) < 0$ for $x \in (\alpha - \epsilon, \alpha + \epsilon)$, which implies that $V_{\alpha-\epsilon} = V_{\alpha+\epsilon}$. Secondly, assume that $\alpha \in [a, b]$ and $A(\alpha) = 0$. By property 1 of a Sturm sequence, $A_1(\alpha - \epsilon)A_2(\alpha - \epsilon) < 0$ and $A_1(\alpha + \epsilon)A_2(\alpha + \epsilon) > 0$; therefore, one sign variation is lost and $V_{\alpha-\epsilon} - V_{\alpha+\epsilon} = 1$. Since $A_1(\alpha + \epsilon)A_2(\alpha + \epsilon) > 0$, $V_{\alpha+\epsilon} = V_\alpha$ and $V_{\alpha-\epsilon} - V_\alpha = 1$. This observation is needed in case $A(a) = 0$ or $A(b) = 0$. Let $\alpha_1, \dots, \alpha_r$ be the distinct roots of $A(x)$ in $(a, b]$. Then

$$V_a - V_b = V_a - V_{\alpha_1} + \sum_{i=1}^{r-1} (V_{\alpha_i} - V_{\alpha_{i+1}}) + V_{\alpha_r} - V_b = r \blacksquare$$

A negative PRS can be used to construct a Sturm sequence.

Definition 22 (Negative PRS) *A negative PRS is a PRS*

$A_1(x), A_2(x), \dots, A_r(x), A_{r+1}(x) = 0$ defined by $e_i A_i(x) = Q_i(x) A_{i+1}(x) + f_i A_{i+2}(x)$, with $e_i f_i < 0$.

Theorem 22 *If $A(x)$ is a squarefree polynomial with real coefficients, then a negative PRS for $A(x)$ and $A'(x)$ is a Sturm sequence for any interval.*

PROOF. We must show that a negative PRS for $A(x)$ and $A'(x)$ satisfies the required properties of a Sturm sequence. Assume $A(\alpha) = 0$. By the mean value theorem, $A(\alpha - \epsilon) = A(\alpha) - A'(\alpha)\epsilon = -A'(\alpha)\epsilon$ for some $x \in (\alpha - \epsilon, \alpha)$ and $A(\alpha + \epsilon) = A(\alpha) + A'(\alpha)\epsilon = A'(\alpha)\epsilon$ for some $x \in (\alpha, \alpha + \epsilon)$. Therefore $A(x)$ and $A'(x)$ have opposite signs for $x \in (\alpha - \epsilon, \alpha)$ and the same signs for $x \in (\alpha, \alpha + \epsilon)$, and hence property 1 is satisfied.

If $A_i(\alpha) = 0$, then $e_{i-1} A_{i-1}(\alpha) = f_{i-1} A_{i+1}(\alpha)$. Since $e_{i-1} f_{i-1} < 0$, $A_{i-1}(\alpha) A_{i+1}(\alpha) < 0$ and property 2 is satisfied. Property 3 follows from the fact that $A(x)$ is squarefree

■

A theorem due to Budan and Fourier gives a similar but weaker test. The theorem of Budan and Fourier gives an upper bound on the number of roots in an interval. This theorem can be stated in two equivalent ways. The first is due to Fourier and the second is due to Budan. We prove a slightly stronger version of Fourier's theorem that is better suited to our algorithmic purposes. The stronger version holds for left-open and right-closed intervals and only requires a subsequence of the derivative sequence in some cases.

Theorem 23 (Fourier's Theorem) *Let $A(x)$ be a polynomial with real coefficients. Assume that for all $x \in [a, b]$ the k -th derivative, $A^{(k)}(x) \neq 0$. Let V_x be the number of sign variations in the derivative sequence $A(x), A^{(1)}(x), \dots, A^{(k)}(x)$. Then $V_a - V_b = r + 2h$ where $h \geq 0$ and r is the number of real roots, multiplicities counted, in the interval $(a, b]$.*

PROOF. The proof is similar to the proof of Sturm's theorem in that the interval $[a, b]$ is partitioned by the roots of a sequence of polynomials (in this case the derivative sequence) and V_x can change only in passing through one of these roots. However, unlike the proof of Sturm's theorem, where V_x changes only in passing through a root of $A(x)$, V_x can change in passing through a root of a derivative, $A^{(i)}(x)$, of $A(x)$.

First assume that $\alpha \in [a, b]$ is a root of multiplicity e of $A(x)$. Using a Taylor series expansion around α , we obtain

$$\begin{aligned} A(\alpha + \epsilon) &= \frac{\epsilon^e A^{(e)}(\alpha)}{e!} + \dots \\ A^{(1)}(\alpha + \epsilon) &= \frac{\epsilon^{e-1} A^{(e)}(\alpha)}{(e-1)!} + \dots \\ &\dots \\ A^{(e-1)}(\alpha + \epsilon) &= \epsilon A^{(e)}(\alpha) + \dots \\ A^{(e)}(\alpha + \epsilon) &= A^{(e)}(\alpha) + \dots \end{aligned}$$

Since for small ϵ the leading term dominates, there are e variations in the sequence $A(\alpha - \epsilon), A^{(1)}(\alpha - \epsilon), \dots, A^{(e)}(\alpha - \epsilon)$ and 0 variations in $A(\alpha + \epsilon), A^{(1)}(\alpha + \epsilon), \dots, A^{(e)}(\alpha + \epsilon)$. Hence, e sign variations are lost as x passes through the root α . Furthermore, by assumption $A^{(r)}(\alpha) \neq 0$ and hence $e \leq r$ so that at least e variations are lost in the

sequence $A(x), A^{(1)}(x), \dots, A^{(r)}(x)$.

Second, assume $\alpha \in [a, b]$ is a root of multiplicity e of $A^{(i)}(x)$ and $A^{(i-1)}(\alpha) \neq 0$. Since $A^{(r)}(\alpha) \neq 0$, $i + e \leq r$. Using the same argument as before, e variations are lost in the subsequence $A^{(i)}(x), \dots, A^{(i+e)}(x)$ as x passes through α . Therefore at least $e - 1 \geq 0$ sign variations are lost in the sequence $A^{(i-1)}(x), \dots, A^{(i+e)}(x)$ as x passes through α . Next we show that the number of variations lost is an even number. Since $A^{(i-1)}(\alpha - \epsilon)$ and $A^{(i-1)}(\alpha + \epsilon)$ have the same signs and $A^{(i+e)}(\alpha - \epsilon)$ and $A^{(i+e)}(\alpha + \epsilon)$ have the same signs, the parity of the number of variations in $A^{(i-1)}(\alpha - \epsilon), \dots, A^{(i+e)}(\alpha - \epsilon)$ is equal to the parity of the number of variations in $A^{(i-1)}(\alpha + \epsilon), \dots, A^{(i+e)}(\alpha + \epsilon)$. Therefore the number of variations lost in the sequence $A^{(i-1)}(x), \dots, A^{(i+e)}(x)$ is a non-negative even number.

The first argument shows that at least r variations are lost as x passes from a to b . The second argument shows that if the number of variations lost exceeds r it must do so by an even number. Finally, since $V_{\alpha+\epsilon} = V_{\alpha}$ the proof holds for left-open and right-closed intervals. ■

Corollary 5 (Budan) *Let $A(x)$ be a polynomial with real coefficients. Let $\text{var}(A(x))$ be the number of variations in the signs of the coefficients of $A(x)$. Let $a < b$ be real numbers. Then $\text{var}(A(x+a)) - \text{var}(A(x+b)) = r + 2h$, where $h \geq 0$ and r is equal to the number of roots of $A(x)$ in $(a, b]$.*

PROOF. Expand $A(x+a)$ and $A(x+b)$ using Taylor series and apply Theorem 23 ■

An important corollary of the theorem of Budan and Fourier is Descartes's rule of signs, which relates the number of positive roots to the number of coefficient sign

variations.

Theorem 24 (Descartes' Rule of Signs) *Let $A(x)$ be a polynomial with real coefficients. Then $\text{var}(A(x)) = r + 2h$, where $h \geq 0$ and r is the number of positive roots of $A(x)$.*

PROOF. Apply Theorem 23 to the interval $(0, \infty)$ and observe that $V_0 = \text{var}(A(x))$ and $V_\infty = 0$ ■

We highlight two special cases which give exact information.

Corollary 6 *If $\text{var}(A(x)) = 0$ then $A(x)$ has no positive real roots.*

Corollary 7 *If $\text{var}(A(x)) = 1$ then $A(x)$ has exactly one positive real root.*

The following theorem gives a partial converse of the first corollary.

Theorem 25 *Let $A(x)$ be a polynomial with real coefficients. If $A(x)$ does not have any roots with positive real parts then $\text{var}(A(x)) = 0$.*

Proof. Assume $a < 0$. Then $\text{var}(x - a) = 0$ and $\text{var}((x - (a + bi))(x - (a - bi))) = \text{var}(x^2 - 2ax + a^2 + b^2) = 0$. The general case follows since the product of two polynomials with zero variations is a polynomial with zero variations ■

The converse of corollary 7 is not true. For example, $A(x) = (x - 1)(x - i)(x + i) = x^3 - x^2 + x - 1$ has one positive real root and three variations. However, the converse is true if the remaining roots are far enough away. The following theorem is due to Vincent. Corollary 8 is due to Akritas and Collins [2].

Theorem 26 (Vincent) *Let $A(x)$ be a squarefree polynomial with real coefficients of degree m . If $A(x)$ has exactly one positive real root and all others are inside a circle of radius*

$$e_m = \left(1 + \frac{1}{m}\right)^{\frac{1}{m-1}} - 1$$

centered at -1 then $\text{var}(A(x)) = 1$.

PROOF. [15] or [48] ■

To apply this result to the coefficient sign variation method, we must consider polynomials with a single root in the interval $(0, 1)$ instead of a single positive root.

Corollary 8 *Let $A(x)$ be a squarefree polynomial with real coefficients of degree m . If $A(x)$ has a single real root in the interval $(0, 1)$ and the remaining roots are outside a circle about the origin of radius m^2 , then $\text{var}((x+1)^m A(1/(x+1))) = 1$.*

PROOF. [15] ■

We present a theorem which strengthens Corollary 8 so that the remaining roots only need to be outside a circle of constant radius.

Theorem 27 *Let $A(x)$ be a squarefree polynomial with real coefficients of degree m . If $A(x)$ has a single root in the interval $(0, 1)$ and all of the remaining real and complex roots are outside the circles of radius 1 centered at $(0, 0)$ and $(1, 0)$, then $\text{var}((x+1)^m A(1/(x+1))) = 1$.*

The proof is taken from the paper of Collins and Johnson [13]. We begin with a series of lemmas.

Lemma 2 *If $\text{var}(A(x)) = 1$ and $B(x) = x + b$, $b \geq 0$ then $\text{var}(A(x)B(x)) = 1$.*

PROOF. Let $A(x) = \sum_{i=0}^m a_i x^i$. We may assume that $a_m > 0$. The lemma is obvious for $b = 0$ so assume that $b > 0$. Let $C(x) = A(x)B(x)$, $C(x) = \sum_{i=0}^{m+1} c_i x^i$. Then $c_{m+1} = a_m > 0$ and $c_0 = a_0 b < 0$. Let a_k be the first positive coefficient of $A(x)$. Then $c_i = a_{i-1} + ba_i \geq 0$ for $k+1 \leq i < m$ and $c_i = a_{i-1} + ba_i \leq 0$ for $1 \leq i < k$. Hence $\text{var}(C(x)) = 1$ regardless of the sign of $c_k = a_{k-1} + ba_k$ ■

We now need two lemmas to handle the case when $A(x)$ has complex conjugate roots.

Lemma 3 *If $\text{var}(A(x)) = 1$ and $B(x) = x^2 + bx + c$, with $b \geq 1$ and $b \geq c > 0$ then $\text{var}(A(x)B(x)) = 1$.*

PROOF. Let $A(x) = \sum_{i \geq 0} a_i x^i$ (with $a_i = 0$ for $i > m$), $C(x) = \sum_{i \geq 0} c_i x^i$. Then $c_i = a_{i-2} + ba_{i-1} + ca_i$ for $i \geq 0$. Assume, without loss of generality, that $a_m > 0$, and let k be the largest integer such that $a_k < 0$. Then obviously $c_i \geq 0$ for $i \geq k+3$ and $c_i \leq 0$ for $i \leq k$. Also $c_{m+2} = a_m > 0$ and, if k is least such that $a_k < 0$ then $c_k < 0$. So $\text{var}(C(x)) \geq 1$ and $\text{var}(C(x)) > 1$ if and only if $c_{k+2} < 0$ and $c_{k+1} > 0$. But $c_{k+2} - c_{k+1} = (a_k + ba_{k+1} + ca_{k+2}) - (a_{k-1} + ba_k + ca_{k+1}) = ca_{k+2} + (b-c)a_{k+1} + (1-b)a_k - a_{k-1}$, a sum of non-negative terms. So $c_{k+2} \geq c_{k+1}$ and hence $c_{k+2} < 0$ with $c_{k+1} > 0$ is impossible ■

Lemma 4 *Let $B(x) = (x - \alpha)(x - \bar{\alpha})$ with α non-real and outside the two circles of radius 1 with centers at $(0,0)$ and $(1,0)$. Let $B^*(x) = (x - \alpha^*)(x - \bar{\alpha}^*) = x^2 + b^*x + c^*$ where $\alpha^* = T_{-1}R(\alpha) = 1/\alpha - 1$. Then $b^* > 1$ and $b^* \geq c^* > 0$.*

PROOF. Let $\alpha^* = e + fi$. Then $b^* = -2e$ and $c^* = e^2 + f^2$. $\alpha \neq 1$ so $1/\alpha \neq 1$, $\alpha^* \neq 0$ and $c^* > 0$. Since α is outside the circle with center $(1, 0)$, $1/\alpha$ is left of the line $x = 1/2$. So $e < -1/2$ and $b^* = -2e > 1$. Since α is outside the circle with center $(0, 0)$, $1/\alpha = \alpha^* + 1$ is inside the same circle. So $(e + 1)^2 + f^2 < 1$, equivalently $e^2 + 2e + f^2 < 0$. But $c^* - b^* = e^2 + f^2 + 2e$ so $b^* > c^*$ ■

We now have all of the machinery needed to prove Theorem 27. We begin by factoring $A(x)$ over the reals as a product of linear and quadratic factors.

$$A(x) = a_m \left\{ \prod_{i=1}^r (x - \alpha_i) \right\} \left\{ \prod_{j=1}^s (x - \beta_j)(x - \overline{\beta_j}) \right\}$$

Let $A_1(x)$ be the product of the linear factors and $A_2(x)$ be the product of the quadratic factors. If exactly one of the α_i is in $(0, 1)$, then applying Lemma 2, r times, the transformed polynomial $T_{-1}R(A_1(x))$ has one variation. Furthermore, by hypothesis each β_j is outside the two circles of radius 1 centered at $(0, 0)$ and $(1, 0)$, hence by Lemmas 3 and 4 applied s times, $\text{var}(T_{-1}R(A(x))) = \text{var}((x + 1)^m A(1/(x + 1))) = 1$. This completes the proof of the theorem ■

Another way of getting exact information from the theorem of Budan and Fourier is to apply the theorem to intervals containing no roots of the derivative. In particular, if for all $x \in [a, b]$ $A'(x) \neq 0$ then $A(x)$ has a root in (a, b) if and only if $A(a)$ and $A(b)$ have opposite signs. This leads to the following theorem.

Theorem 28 (Root Counting Based on Rolle's Theorem) *Let $A(x)$ be a square-free polynomial with real coefficients. Let $\alpha_1, \dots, \alpha_r$ be the real roots of $A'(x)$. Then the the number of sign variations in $A(-\infty), A(\alpha_1), \dots, A(\alpha_r), A(\infty)$ is equal to then number of real roots of $A(x)$.*

This theorem can be applied inductively to the derivative sequence of a polynomial. The root isolation algorithms in Section 4.3 use this inductive process and Theorem 28 to recursively obtain isolating intervals for a polynomial from isolating intervals of its derivative.

Induction using the derivative sequence can also be used to give an alternative proof of the theorem of Budan and Fourier (Theorem 23). Some ideas used in this proof are used in the derivation of a root isolation algorithm in Section 4.3. We begin with a lemma needed to carry out the induction.

Lemma 5 *Let k be the first positive integer such that $A^{(k)}(a) \neq 0$ and assume that $A^{(1)}(b) \neq 0$.*

1. *Assume for all $x \in (a, b]$, $A(x) \neq 0$. Also assume $A(a) = 0$ or $A(a)$ and $A^{(k)}(a)$ have the same sign, and assume $A(b)$ and $A^{(1)}(b)$ have opposite signs. Then there exists a root of $A^{(1)}(x)$ in the interval $(a, b]$.*
2. *Assume $A(\alpha) = 0$ for some $\alpha \in (a, b]$. Assume $A(a)$ and $A^{(k)}(a)$ have the same sign. Then there exists a root of $A^{(1)}(x)$ in the interval (a, α) .*
3. *Assume $A(\alpha) = 0$ for some $\alpha \in (a, b]$. Assume $A(b)$ and $A^{(1)}(b)$ have opposite signs. Then there exists a root of $A^{(1)}(x)$ in the interval (α, b) .*

PROOF. We only prove part 1 since the other two parts can be proven similarly. Without loss of generality we can assume $A(x) > 0$ for $x \in (a, b]$. Using a Taylor series expansion

$$A(a + \epsilon) = A(a) + \frac{A^{(k)}(a)}{k!} \epsilon^k + \dots$$

$$A(b - \epsilon) = A(b) - A^{(1)}(b)\epsilon + \dots$$

Since, by hypothesis, $A(a) = 0$ or $A(a)$ and $A^{(k)}(a)$ have the same sign, $A(a + \epsilon) > A(a)$ for small ϵ . Also, since $A(b)$ and $A^{(1)}(b)$ have opposite signs, $A(b - \epsilon) > A(b)$ for small ϵ . Therefore the maximum value of $A(x)$ for $x \in [a, b]$ is obtained at $\gamma \in (a, b)$ and $A^{(1)}(\gamma) = 0$ ■

Lemma 6 *Let $A(x)$ be a polynomial with real coefficients. Let V_x be the number of sign variations in the derivative sequence $A(x), A^{(1)}(x), \dots, A^{(m)}(x)$. Assume $a < b$. Then $V_a - V_b \geq r$, where r is the number of roots, multiplicities counted, of $A(x)$ in $(a, b]$.*

PROOF. The proof is by induction on m , the degree of $A(x)$. If $m = 0$, the lemma is trivially true. Assume that the lemma is true for all polynomials of degree less than $m \geq 1$.

Let r' be the number of real roots of $A^{(1)}(x)$ in the interval $(a, b]$ and V'_x be the number of variations of the derivative sequence of $A^{(1)}(x)$ evaluated at x . By induction $V'_a - V'_b \geq r'$. We need to relate $V_a - V_b$ to $V'_a - V'_b$ and r to r' .

Either $V_x = V'_x$ or $V_x = V'_x + 1$. Let $A^{(k)}(x)$ be the first derivative that does not vanish at x . If either $A(x) = 0$ or $A(x)$ and $A^{(k)}(x)$ have the same signs, then $V_x = V'_x$. If $A(x)$ and $A^{(k)}(x)$ have opposite signs, then $V_x = V'_x + 1$. Since $V_x = V'_x$ or $V_x = V'_x + 1$, $V_a - V_b = V'_a - V'_b$, $V'_a - V'_b + 1$, or $V'_a - V'_b - 1$.

If $r = 0$, all we need to show is that $V_a - V_b \geq 0$. By induction $V'_a - V'_b \geq 0$. Therefore, $V_a - V_b$ is necessarily greater than or equal to zero unless $V_a - V_b =$

$V'_a - V'_b - 1$. If $V_a - V_b = V'_a - V'_b - 1$, then $V_a = V'_a$ and $V_b = V'_b + 1$ and by the discussion above either $A^{(1)}(b) = 0$ or the hypotheses of part 1 of Lemma 5 are satisfied. In either case, $r' \geq 1$, hence $V_a - V_b \geq V'_a - V'_b - 1 \geq r' - 1 \geq 0$ in this case also.

Assume there are $r \geq 1$ roots, $\alpha_1 \geq \dots \geq \alpha_r$, of $A(x)$ in the interval $(a, b]$. By Rolle's theorem $r' \geq r - 1$. If $V_a = V'_a$, either $A(a) = 0$ or $A(a)$ and $A^{(k)}(a)$ have the same signs, where k is the first positive integer such that $A^{(k)}(a) \neq 0$. If $A(a) = 0$, Rolle's theorem implies there exists a root of $A^{(1)}(x)$ in (a, α_1) , and if $A(a)$ and $A^{(k)}(a)$ have the same signs, part 2 of Lemma 5 implies there exists a root of $A^{(1)}(x)$ in (a, α_1) . Therefore if $V_a = V'_a$, then $r' \geq r$.

If $V_b = V'_b + 1$, either $A^{(1)}(b) = 0$ or $A(b)$ and $A^{(1)}(b)$ have opposite signs. If $A^{(1)}(b) = 0$, then $r' \geq r$, and if $A(b)$ and $A^{(1)}(b)$ have opposite signs, part 3 of Lemma 5 implies that there exists a root of $A^{(1)}(x)$ in (α_r, b) . Therefore if $V_b = V'_b + 1$, then $r' \geq r$. If $V_a = V'_a$ and $V_b = V'_b + 1$ there are roots of the derivative in (a, α_1) and (α_r, b) and $r' \geq r + 1$.

To complete the induction, four cases must be considered. First, suppose $V_a = V'_a$ and $V_b = V'_b$. Then $V_a - V_b = V'_a - V'_b \geq r' \geq r$. Second, suppose $V_a = V'_a$ and $V_b = V'_b + 1$. Then $V_a - V_b = V'_a - V'_b - 1 \geq r' - 1 \geq r$. Third, suppose $V_a = V'_a + 1$ and $V_b = V'_b$. Then $V_a - V_b = V'_a - V'_b + 1 \geq r' + 1 \geq r$. Finally, suppose $V_a = V'_a + 1$ and $V_b = V'_b + 1$. Then $V_a - V_b = V'_a - V'_b \geq r' \geq r$ ■

The proof of Fourier's theorem is completed using this lemma and the observation that $V_a - V_b$ and the number of roots in the interval $(a, b]$ have the same parity.

CHAPTER III

Auxiliary Algorithms

This chapter discusses some auxiliary algorithms that are needed for integral polynomial real root isolation. Special attention is given to the implementation of efficient programs for these operations. Tables of empirical computing times are given to help estimate the computing times of higher level algorithms which use these operations as subalgorithms. The tables also show the importance of some of the efficiency considerations that are discussed.

Section 3.1 introduces several types of intervals such as binary rational intervals and standard intervals. Use of these special intervals significantly reduce the time required to perform important interval operations such as bisection and polynomial evaluation at the endpoints. Sections 3.2 and 3.3 present efficient algorithms for performing integral polynomial transformations and evaluating an integral polynomial at a rational number. Section 3.4 shows how to implement the root bound theorems in Section 2.2.1. The chapter concludes with a review of algorithms for computing the gcd of two integral polynomials.

3.1 Interval Operations

Definition 23 (Interval) *The open interval $(a, b) = \{x : a < x < b\}$. The left-open*

and right-closed interval $(a, b] = \{x : a < x \leq b\}$. The closed interval $[a, b] = \{x : a \leq x \leq b\}$.

Definition 24 (Isolating Interval and Strong Isolating Interval) *An isolating interval for a polynomial $A(x)$ is an interval that contains exactly one distinct real root of $A(x)$. An isolating interval may contain more than one real root if the root is of multiplicity greater than one. A strong isolating interval is an interval whose closure is an isolating interval.*

We will only consider intervals with rational endpoints. A rational number $r = a/b$ is represented by the list (a, b) where a and $b \geq 1$ are integers and $\gcd(a, b) = 1$. An integer $a = \sum_{i=0}^m a_i \beta^i$ is represented by the list of β -digits $(a_m, \dots, a_0)$. The radix $\beta = 2^k$ is chosen to be a power of 2. For more information on the representation of integers and rational numbers and algorithms for performing arithmetic with them see [11, 16, 17].

Since β is a power of two, it is advantageous to use binary rational numbers, instead of arbitrary rational numbers, as endpoints. Furthermore binary rational numbers are particularly well suited to interval bisection.

Definition 25 (Binary Rational Number) *A binary rational number is a rational number whose denominator is a power of 2. The binary rational numbers form an overring of the integers denoted by $Z[1/2]$.*

Arithmetic with binary rational numbers is particularly simple to perform on a

binary computer.

$$\frac{a}{2^e} \cdot \frac{b}{2^f} = \frac{ab}{2^{e+f}} \text{ (if } e = 0 \text{ or } f = 0 \text{ this may not be reduced)}$$

$$\frac{a}{2^e} + \frac{b}{2^f} = \begin{cases} \frac{a+b}{2^e} & \text{if } e = f. \text{ (this may not be reduced)} \\ \frac{a+b2^{e-f}}{2^e} & \text{if } e > f. \\ \frac{a2^{f-e}+b}{2^f} & \text{if } e < f. \end{cases}$$

Several auxiliary algorithms are needed to perform arithmetic with binary rational numbers. These algorithms are listed in Figures 1, 2, and 3.

$$n \leftarrow \mathbf{IORD2}(a)$$

[Integer, order of 2. a is a non-zero integer. n is the largest integer such that 2^n divides a .]

1. [Count low order β -digits.] $k \leftarrow$ smallest integer such that $a_k \neq 0$.
2. [Count low order bits.] $r \leftarrow$ number of trailing zero bits in a_k ; $n \leftarrow k\zeta + r$ ■

Figure 1: **IORD2** Integer, Order of 2

$$n \leftarrow \mathbf{ILOG2}(A)$$

[Integer logarithm, base 2. A is an integer. If $A = 0$ then $n = 0$. Otherwise $n = \lfloor \log_2(|A|) \rfloor + 1$, a β -digit.]

1. if $A = 0$ then { $n \leftarrow 0$; return }.
2. $m \leftarrow L_\beta(A)$; $r \leftarrow$ number of bits in $|a_{m-1}|$; $n \leftarrow m\zeta + r$ ■

Figure 2: **ILOG2** Integer Logarithm, Base 2

Theorem 29 *The computing times of $\mathbf{IORD2}(A)$ and $\mathbf{ILOG2}(A)$ are dominated by $L(A)$. The computing time of $\mathbf{ITRUNC}(A, n)$ is dominated by $L(A) + n$.*

$$B \leftarrow \text{ITRUNC}(A, n)$$

[Integer truncation. A is an integer. n is a β -integer. $B = \lfloor A/2^n \rfloor$.]

1. [$A = 0$ or $n = 0$.] if $A = 0 \vee n = 0$ then { $B \leftarrow A$; return }.
2. [Let $|n| = q\zeta + r$, $0 \leq r < \zeta$.] $\text{QREM}(|n|, \zeta; q, r)$.
3. [$n > 0$. Delete q trailing digits and shift remaining digits.] $B \leftarrow (A2^{-q\zeta})2^{-r}$.
4. [$n < 0$. Shift digits of A by r bits and append q trailing zero digits.] $B \leftarrow (A2^r)2^{q\zeta}$ ■

Figure 3: ITRUNC Integer Truncation

Definition 26 (Binary Rational Interval) *A binary rational interval is an interval whose endpoints are binary rational numbers. A binary rational interval can be written in the form $(\frac{a}{2^c}, \frac{a+l}{2^c})$.*

Lemma 7 (Closure Under Bisection) *The midpoint of a binary rational interval is a binary rational number. Moreover, if $I = (\frac{a}{2^c}, \frac{a+l}{2^c})$, then after bisection, the left subinterval $I_1 = (\frac{2a}{2^{c+1}}, \frac{2a+l}{2^{c+1}})$ and the right subinterval $I_2 = (\frac{2a+l}{2^{c+1}}, \frac{2a+2l}{2^{c+1}})$.*

It is important for the endpoints of an interval to be as small as possible. This involves minimizing the size of the endpoints compared to the width of the interval.

Definition 27 (Interval Width) *Let I be an interval with endpoints a and b . The width $W(I) = b - a$.*

Definition 28 (Interval Absolute Value) *Let I be an interval with endpoints a and b . The absolute value of I , $|I| = \max(|a|, |b|)$.*

Definition 29 (Length of a Rational Number) *Let $r = e/f$ be a rational number. The length of r , $L(r) = L(e) + L(f) \sim \max(L(e), L(f))$.*

Definition 30 (Interval Size) *Let I be an interval with endpoints a and b . The size of I , $L(I) = L(a) + L(b) \sim \max(L(a), L(b))$.*

The algorithm **RIB**, listed in Figure 4, in many cases produces a smaller bisection point than the midpoint. If $2^h \leq W(I) < 2^{h+1}$, **RIB** chooses the smallest integer multiple of 2^h contained in I as the bisection point. For example, if $I = (3/8, 3/4)$, the midpoint is $9/16$ and the bisection point produced by **RIB** is $1/2$. The algorithm **RIB** is due to Collins.

$$c \leftarrow \mathbf{RIB}(I)$$

[Rational interval bisection. $I = (a, b)$ is a binary rational interval. c is a binary rational number with $r < c < s$, defined as follows. Let $2^h \leq W(I) < 2^{h+1}$. If $W(I) = 2^h$, then $c = (a + b)/2 = a + 2^{h-1}$, otherwise c is equal to the smallest integer multiple of 2^h in I .]

1. [Compute h and c .] $a \leftarrow \mathbf{LeftEndpoint}(I)$; $b \leftarrow \mathbf{RightEndpoint}(I)$; $l \leftarrow \mathbf{num}(W(I))$; $e \leftarrow \mathbf{IORD2}(\mathbf{den}(W(I)))$;
 if $l = 1$ then {
 $c \leftarrow (a + b)/2$; return }
 else {
 $h_1 \leftarrow \mathbf{ILOG2}(l) - 1$; $h \leftarrow h_1 - e$; $c \leftarrow \mathbf{ITRUNC}(a, h)$ } ■

Figure 4: **RIB** Rational Interval Bisection

Definition 31 (Standard Interval) *A standard interval is a binary rational interval of the form $(\frac{a}{2^e}, \frac{a+1}{2^e})$.*

A standard interval has length equal to a power of two. Bisection of a standard interval, using **RIB**, is the same as computing the midpoint. Furthermore, the size of a standard interval compared to the length of a standard interval is small. Most importantly, the midpoint of a standard interval has at most one extra bit in the numerator and denominator.

RIB can be used to refine a binary rational isolating interval $I = (a, b)$ for a polynomial $A(x)$. At each step in the refinement process, **RIB** partitions I into two subintervals and the subinterval containing the root of $A(x)$ is chosen. Repeated application of **RIB** in this manner will eventually produce a standard interval. The algorithm **IPSIFI**, listed in Figure 5, uses this idea to obtain a standard isolating interval from a binary rational isolating interval. This refinement process can continue until the width of the isolating interval is as small as desired.

It is clear that **IPSIFI** will eventually produce a standard interval. Since, if $I = (a/2^e, (a + l)/2^e)$, the bisection point produced by **RIB** is a multiple of 2^e and the width of the chosen subinterval will be equal to $l'/2^e$ with $l' < l$. The following lemma can be used to bound the number of required bisections.

Lemma 8 *Let $I = (a, b)$ be a binary rational interval and let $c = \mathbf{RIB}(I)$. Let $I_1 = (a, c)$ and $I_2 = (c, b)$ and $c_1 = \mathbf{RIB}(I_1)$ and $c_2 = \mathbf{RIB}(I_2)$. Then (c_1, c) and (c, c_2) are standard intervals. Moreover, $W((c_1, c)) \geq 1/2W(I_1)$ and $W((c, c_2)) \geq 1/2W(I_2)$*

PROOF. Let $2^h \leq W(I) < 2^{h+1}$ and $2^{h_1} \leq W(I_1) < 2^{h_1+1}$. By the construction of **RIB**, c is the smallest integral multiple of 2^h in I and c_1 is the smallest integral multiple of 2^{h_1} in I_1 . Since I_1 is a subinterval of I , 2^{h_1} is a divisor of 2^h and c is an

$$J \leftarrow \mathbf{IPSIFI}(A(x), I)$$

[Integral polynomial standard isolating interval from isolating interval. I is an interval with binary rational endpoints, which is either left-open and right-closed or a one-point interval. $A(x)$ is a univariate integral polynomial which has a unique root α of odd multiplicity in I . If I is a one-point interval, then $J = I$. If I is left-open and right closed, then J is either a standard left-open and right-closed subinterval of I containing α , or if α is a binary rational number, J may possibly instead be the one-point interval (α, α) .]

1. [Initialize.] $a \leftarrow \text{LeftEndpoint}(I)$; $b \leftarrow \text{RightEndpoint}(I)$; $t \leftarrow \text{sign}(A(b))$; if $t = 0$ then { $J \leftarrow (b, b)$; return } else $J \leftarrow I$.
2. [Bisect.]
 while J is not Standard do {
 $c \leftarrow \mathbf{RIB}(J)$; $s \leftarrow \text{sign}(A(c))$;
 if $s = 0$ then {
 $J \leftarrow (c, c)$; return }
 else if $st \leq 0$ then { $J = (c, b)$ }
 else { $J \leftarrow (a, c)$; $t \leftarrow s$ } } ■

Figure 5: **IPSIFI** Integral Polynomial Standard Isolating Interval from Interval

integral multiple of 2^{h_1} . Since $W(I_1) < 2^{h_1+1}$ and c is an integral multiple of 2^{h_1} there is exactly one integral multiple of 2^{h_1} in I_1 which is equal to c_1 and the next integral multiple of 2^{h_1} is c . Therefore, $c = (t+1)2^{h_1}$ and $c_1 = t2^{h_1}$ and (c_1, c) is a standard interval. Furthermore, $W((c_1, c)) = 2^{h_1} \geq 1/2W(I_1)$. The same argument applies for (c, c_2) ■

Theorem 30 (Maximum Number of Bisections Needed by IPSIFI.) *Let $I = (a/2^e, (a+l)/2^e)$ be a binary rational interval. The maximum number of bisections required by IPSIFI is less than or equal to $\lfloor \log_2(l) \rfloor + 1$.*

PROOF. Let $J = (a, b)$ be the subinterval of I obtained after one bisection and let J_1 be the interval obtained after the second bisection. By Lemma 8 either J_1 is standard or $W(J_1) \leq 1/2W(J)$. Therefore, $W(J_1) \leq (l/2)/2^e$. Repeatedly applying this argument implies that after $\lfloor \log_2(l) \rfloor$ bisections a standard subinterval of width $1/2^e$ will be obtained. ■

This bound can be obtained. Let $I = (1/2^n, 1)$ and suppose **IPSIFI** repeatedly chooses the left subinterval. After j bisections the subinterval is equal to $(1/2^n, 1/2^j)$ and the process terminates in $n - 1$ steps with the standard interval $(1/2^n, 1/2^{n-1})$. Usually **IPSIFI** needs far fewer bisections than the maximum amount in Theorem 30.

Theorem 31 (Expected Number of Bisections Needed by IPSIFI) *Let I be a binary rational isolating interval for $A(x)$. Assume that the isolated root of $A(x)$ is uniformly distributed in I . Then the expected number of bisections needed by IPSIFI is less than or equal to 5.*

PROOF.

Let $J = (a, b)$ be the subinterval of I obtained after one bisection. Let $c = \text{RIB}(J)$ and $J_1 = (a, c)$ and $J_2 = (c, b)$. By Lemma 8 one of these intervals is standard and the length of the standard interval is greater than or equal to half of the length of J . Since the root of $A(x)$ is assumed to be uniformly distributed in I the probability that a subinterval J_i is selected after bisection is equal to $W(J_i)/W(J)$. Therefore the probability that the standard interval is selected is greater than or equal to $1/2$.

Let p_k be the probability that exactly k bisections of J are needed to obtain a standard interval. This probability is less than or equal to the probability that k or more bisections are needed, which is less than or equal to 2^{-k+1} . Therefore, the expected number of bisections is equal to

$$E = 1 + \sum_{k=1}^{\infty} p_k k \leq 1 + \sum_{k=1}^{\infty} k/2^{k-1} = 5 \blacksquare$$

An interval containing a single root of a polynomial $A(x)$ and a single root of a polynomial $B(x)$ can be refined into two isolating intervals: one for $A(x)$ and one for $B(x)$. The initial interval is repeatedly bisected, always choosing the subinterval containing the root of $A(x)$, until it does not contain the root of $B(x)$. This algorithm is used in the root isolation algorithm in Section 4.3. The specifications for the algorithm that is required is listed in Figure 6.

3.2 Polynomial Transformations

This section presents algorithms for computing polynomial reciprocal transformations, polynomial homothetic transformations, and polynomial translations. A com-

$$\text{IPRRS}(A(x), s_1, t_1, B(x), s_2, t_2, I; I^*, s_1^*, t_1^*, s_2^*, t_2^*)$$

[Integral polynomial real root separate. Inputs: $A(x)$ is a squarefree integral polynomials containing a single root, α , in the interval binary rational interval $I = (a, b]$. $s_1 = \text{sign}(A(a))$. $t_1 = \text{sign}(A(b))$. $B(x)$ is an integral polynomial for which either $B'(x) \neq 0$ for all $x \in I$ or $B(x)$ has a unique root of odd multiplicity in I . $s_2 = \text{sign}(B(a))$. $t_2 = \text{sign}(B(b))$. Outputs: $I^* = (a^*, b^*]$ is an isolating interval for α containing no roots of $B(x)$. $s_1^* = \text{sign}(A(a^*))$. $t_1^* = \text{sign}(A(b^*))$. $s_2^* = \text{sign}(B(a^*))$. $t_2^* = \text{sign}(B(b^*))$.]

Figure 6: **IPRRS** Integral Polynomial Real Root Separate

puting time bound is derived for each algorithm and a table of empirical computing times is given.

Let $A(x)$ be a polynomial of degree m . The reciprocal transformation can be viewed as a linear transformation on the vector space of polynomials of degree m .

$R: x^k \longrightarrow x^m x^{-k} = x^{m-k}$. Using the basis $\{x^m, \dots, x, 1\}$,

$$R = \begin{pmatrix} & & 1 \\ & \ddots & \\ 1 & & \end{pmatrix},$$

and we see that R inverts the coefficients of $A(x)$. The algorithm **PRT**, listed in Figure 7, implements this transformation.

Theorem 32 (Computing Time of PRT) *If $\deg(A(x)) = m$ and $A(x)$ is represented as a list of coefficients, then **PRT** computes $R(A(x))$ in time dominated by m .*

The homothetic transformation H_a induces the linear transformation: $H_a: x^k \longrightarrow$

$$B(x) \leftarrow \mathbf{PRT}(A(x))$$

[Polynomial reciprocal transformation. $A(x) = \sum_{i=0}^m a_i x^i \neq 0$ is a polynomial of degree m . $B(x) = \sum_{i=0}^m b_i x^i = x^m A(1/x)$.]

1. for $i = 0, \dots, m$ do
 $b_i = a_{m-i}$ ■

Figure 7: **PRT** Polynomial Reciprocal Transformation

$a^m(x/a)^m = a^{m-k}x^k$. Using the basis $\{x^m, \dots, x, 1\}$,

$$H_a = \begin{pmatrix} 1 & & & \\ & a & & \\ & & \ddots & \\ & & & a^m \end{pmatrix}.$$

The inverse homothetic transformation $H_{1/a} = RH_aR$ maps x^k to $(ax)^k$. The algorithm **IUPHT**, listed in Figure 8, implements integer homothetic transformations.

$$B(x) \leftarrow \mathbf{IUPHT}(A(x), a)$$

[Integral univariate polynomial homothetic transformation. $A(x) = \sum_{i=0}^m a_i x^i$ is an integral polynomial of degree m . a is an integer. $B(x) = \sum_{i=0}^m b_i x^i = A(ax)$.]

1. [Initialize and compute homothetic transformation.] $C \leftarrow 1$; $b_0 \leftarrow a_0$;
 for $i = 1, \dots, m$ do {
 $C \leftarrow a \cdot C$; $b_i \leftarrow a_i C$ } ■

Figure 8: **IUPHT** Integral Univariate Polynomial Homothetic Transformation

Polynomial negation, $A^*(x) = A(-x)$, is a special homothetic transformation and is implemented by **IUPNT** (see Figure 9).

$$B \leftarrow \text{IUPNT}(A(x))$$

[Integral univariate polynomial negative transformation. $A(x)$ is an integral polynomial of degree m . $B(x) = A(-x)$.]

1. [Initialize and compute homothetic transformation.] for $i=0, \dots, m$ do {
 $b_i \leftarrow -1^i a_i$ } ■

Figure 9: **IUPNT** Integral Univariate Polynomial Negative Transformation

Theorem 33 (Computing Time of IUPHT) *If $\deg(A(x)) = m$ and a is an integer, then IUPHT computes $A(ax)$ in time dominated by $m^2\{L(|a|)L(|A(x)|_\infty) + L(|a|)^2\}$.*

PROOF. The computing time is dominated by

$$\begin{aligned} \sum_{i=1}^m L(|a|^i)L(|a_i|) + \sum_{i=1}^{m-1} L(|a|)^i L(|a|) &\preceq (L(|A(x)|_\infty)L(|a|) + L(|a|)^2) \sum_{i=0}^m i \\ &\preceq m^2(L(|a|)L(|A(x)|_\infty) + L(|a|)^2) \blacksquare \end{aligned}$$

The algorithm **IUPBHT** implements binary homothetic transformations (a is a power of two) using **ITRUNC**. Figure 10 lists the specification for **IUPBHT**.

$$B(x) \leftarrow \text{IUPBHT}(A(x), k)$$

[Integral univariate polynomial binary homothetic transformation. $A(x) = \sum_{i=0}^m a_i x^i$ is an integral polynomial of degree m . k is an integer. If $k \geq 0$, $B(x) = A(2^k x)$ else $B(x) = 2^{km} A(2^k x)$.]

Figure 10: **IUPBHT** Integral Univariate Polynomial Binary Homothetic Transformation

Theorem 34 (Computing Time of IUPBHT) *If $\deg(A(x)) = m$, then IUPBHT computes $A(2^k x)$ in time dominated by $mL(|A(x)|_\infty) + km^2$.*

PROOF. The computing time is dominated by

$$\sum_{i=0}^m (L(|a_i|) + ki) \preceq mL(|A(x)|_\infty) + km^2 \blacksquare \quad (3.1)$$

The polynomial translation T_{-b} induces the linear transformation: $T_{-b} : x^k \longrightarrow (x + b)^k$. Using the basis $\{x^m, \dots, x, 1\}$,

$$T_{-b} = \begin{pmatrix} \begin{pmatrix} m \\ m \end{pmatrix} \\ \begin{pmatrix} m \\ m-1 \end{pmatrix} b & \begin{pmatrix} m-1 \\ m-1 \end{pmatrix} \\ \vdots & \vdots & \ddots \\ \begin{pmatrix} m \\ 0 \end{pmatrix} b^m & \begin{pmatrix} m-1 \\ 0 \end{pmatrix} b^{m-1} & \dots & \begin{pmatrix} 0 \\ 0 \end{pmatrix} \end{pmatrix}.$$

Polynomial translation can be computed using Horner's rule:

$$A_0(x) = a_m$$

$$A_i(x) = A_{i-1}(x)(x + b) + a_{m-i}$$

The algorithm **IUPTR**, listed in Figure 11, implements integral polynomial translation using Horner's method.

IUPTR1($A(x)$) is the special case of **IUPTR**($A(x), b$) where b is assumed to be one. In this case no multiplications are required. **IUPTR1** has been improved further by storing the coefficients in an array. The coefficients are padded with extra zeroes to ensure enough space for the coefficients of the translated polynomial. The bound in Theorem 20 can be used to compute how much space is required.

$$B(x) \leftarrow \mathbf{IUPTR}(A(x), b)$$

[Integral univariate polynomial translation. $A(x) = \sum_{i=0}^m a_i x^i$ is a integral polynomial. b is an integer. $B(x) = A(x + b)$.]

1. [Compute translation using Horner's rule.] $B(x) \leftarrow a_m$;
 for $i = 0, \dots, m$ do
 $B(x) \leftarrow B(x)(x + b) + a_{m-i}$ ■

Figure 11: **IUPTR** Integral Univariate Polynomial Translation

Theorem 35 (Computing Time of IUPTR and IUPTR1)

Assume $A(x) = \sum_{i=0}^m a_i x^i$ is an integral polynomial of degree m and b is an integer.

1. **IUPTR**($A(x), b$), computes $A(x + b)$ in time dominated by
 $m^3 L(|b|)^2 + m^2 L(|b|) L(|A(x)|_\infty)$.
2. **IUPTR1**($A(x)$), computed $A(x + 1)$ in time dominated by $m^3 + m^2 L(|A(x)|_\infty)$.

PROOF. By Theorem 20, $|B(x)|_\infty \leq |A(x)|_\infty (1 + |b|)^{m+1}$. Therefore the each multiplication is dominated by $L(|b|)(L(|A(x)|_\infty) + mL(|b|)$ and each addition is dominated by $L(|A(x)|_\infty) + mL(|b|)$. Since **IUPTR** performs m^2 coefficient multiplications and additions the total time is dominated by. $m^2 L(|b|)(L(|A(x)|_\infty) + m^3 L(|b|))$ ■

Table 1 reports computing times for **IUPHT**($A(x), 2^k$), **IUPBHT**($A(x), k$), and **IUPTR**($A(x), 2^k$). Table 2 reports computing times for **IUPTR**($A(x), 1$), **IUPTR1**($A(x)$), and **IUPBHT**($A(x), 1$). The polynomial $A(x)$ is an integral polynomial of degree m with random coefficients between -2^d and 2^d . All times are in milliseconds and are averaged over 100 iterations. These tables show the importance of using

binary homothetic transformations instead of arbitrary homothetic transformations and specializing polynomial translation to translation by one.

3.3 Polynomial Evaluation

Several algorithms for evaluating an integral polynomial at a rational number, based on Horner's method, are presented and compared.

The first algorithm uses rational arithmetic to evaluate $A(r)$, where $A(x)$ is a rational polynomial and r is a rational number. Horner's method is described by the recursive formula:

$$\begin{aligned} b_0 &= a_m \\ b_i &= b_{i-1}r + a_{m-i}. \end{aligned}$$

The algorithm **RUPRE**, listed in Figure 12, implements this scheme.

$$b \leftarrow \mathbf{RUPRE}(A(x), r)$$

[Rational univariate polynomial rational evaluation. $A(x) = \sum_{i=0}^m a_i x^i$ is a rational polynomial and r is a rational number. $b = A(r)$.]

1. [Evaluate $A(r)$ using Horner's method.] $b \leftarrow a_m$;
 for $i = 1, \dots, m$ do
 $b \leftarrow br + a_{m-i}$ ■

Figure 12: **RUPRE** Rational Univariate Polynomial Rational Evaluation

If $A(x)$ is an integral polynomial, rational arithmetic can be avoided. If $r = e/f$, then $\bar{b} = f^m A(e/f)$ is an integer and can be computed with integral arithmetic using

Table 1: Computing Times (in ms) of IUPHT, IUPBHT, and IUPTR

m	d	k	IUPHT($A(x), 2^k$)	IUPBHT($A(x), k$)	IUPTR($A(x), 2^k$)
10	10	10	0.84	1.33	2.83
10	10	155	8.33	2.17	31.83
10	10	300	15.66	2.68	55.83
10	155	10	1.83	2.17	7.50
10	155	155	40.00	4.00	42.00
10	155	300	75.01	4.16	74.00
10	300	10	2.83	3.16	11.67
10	300	155	65.84	4.50	49.84
10	300	300	126.33	5.33	71.84
20	10	10	2.00	3.00	15.33
20	10	155	33.16	5.50	232.18
20	10	300	61.82	8.17	421.84
20	155	10	8.00	5.83	33.16
20	155	155	150.17	8.34	252.99
20	155	300	285.83	11.01	492.34
20	300	10	6.49	6.17	47.17
20	300	155	251.67	8.66	284.97
20	300	300	490.33	11.51	486.70
30	10	10	4.16	4.17	41.82
30	10	155	71.17	10.51	764.96
30	10	300	138.17	15.82	1457.10
30	155	10	20.17	9.33	78.50
30	155	155	331.00	14.51	848.54
30	155	300	632.17	20.18	1551.84
30	300	10	14.33	10.33	114.33
30	300	155	550.17	15.16	857.67
30	300	300	1065.84	20.99	1529.94

Table 2: Computing Times (in ms) of IUPTR, IUPTR1, and IUPBHT

m	d	IUPTR($A(x), 1$)	IUPTR1($A(x)$)	IUPBHT($A(x), 1$)
10	10	0.83	0.50	1.17
10	155	6.67	1.16	2.67
10	300	10.82	1.84	3.33
10	445	15.17	2.66	3.50
20	10	2.83	1.17	2.34
20	155	24.50	3.16	4.51
20	300	40.51	4.83	5.83
20	445	59.66	7.00	6.66
30	10	9.67	2.33	3.00
30	155	55.18	6.00	7.33
30	300	91.33	9.84	8.50
30	445	129.32	13.67	9.17
40	10	23.33	3.17	4.67
40	155	99.00	9.83	10.34
40	300	160.83	15.33	12.16
40	445	224.83	21.33	13.17
50	10	42.33	6.16	6.00
50	155	154.99	15.00	12.00
50	300	254.17	24.33	15.83
50	445	351.60	32.83	15.51

the following modification to Horner's method:

$$\begin{aligned}\bar{b}_0 &= a_m \\ \bar{b}_i &= \bar{b}_{i-1}e + a_{m-i}f^i.\end{aligned}$$

The algorithm **IUPREI**, listed in Figure 13, computes $f^m A(e/f)$ using this version of Horner's method.

$$\bar{b} \leftarrow \mathbf{IUPREI}(A(x), r)$$

[Integral univariate polynomial rational evaluation, integral part. $A(x) = \sum_{i=0}^m$ is a integral polynomial. $r = e/f$ is a rational number. $\bar{b} = f^m A(e/f)$.]

1. [Evaluate $f^m A(e/f)$ using Horner's method.] $\bar{b} \leftarrow a_m$; $F \leftarrow 1$;
 for $i = 1, \dots, m$ do
 $F \leftarrow fF$; $\bar{b} \leftarrow \bar{b}e + a_{m-i}F$ ■

Figure 13: **IUPREI** Integral Univariate Polynomial Rational Evaluation Integral Part

If only the sign of $A(r)$ is desired, computing $f^m A(e/f)$ suffices since $\text{sign}(A(e/f)) = \text{sign}(f^m A(e/f))$. If the actual value of $A(e/f)$ is needed, $A(e/f)$ is obtained by reducing $f^m A(e/f)/f^m$ to lowest terms. The algorithm **IUPRE1** first computes $\bar{b} = f^m A(e/f)$ and then reduces $\bar{b}/f^m$ to lowest terms. This reduction involves a costly gcd computation; however, this can be remedied with the aid of the following theorem.

Theorem 36 $\gcd(a, b^m) = \prod_{i=1}^m g_i$, where $g_i = \gcd(a / \prod_{j=1}^{i-1} g_j, b)$.

PROOF. By induction on m using the identity

$$\gcd(a, b^m) = \gcd(a, b) \gcd(a / \gcd(a, b), b^{m-1}) \blacksquare$$

The algorithm **IUPRE** incorporates this improvement. Furthermore, if $g_i = 1$ for any i , the algorithm can stop. Since typically $g_i = 1$ for all but the first few i , the improvement in computing time should be proportional to m .

All four of the evaluation algorithms **RUPRE**, **IUPREI**, **IUPRE1**, and **IUPRE** have the same dominance relation for their computing times; however, the actual computing times are significantly different (see Table 3).

Theorem 37 (Computing Time of **RUPRE, **IUPREI**, **IUPRE1**, **IUPRE**)**

*Let $A(x)$ be an integral polynomial of degree m , and let $r = e/f$ be a rational number. The three algorithms **RUPRE**, **IUPRE1**, and **IUPRE** compute $A(r)$ in time dominated by $mL(|A(x)|_\infty)L(r) + m^2L(r)^2$, and **IPUREI** computes $f^m A(e/f)$ in time dominated by $mL(|A(x)|_\infty)L(r) + m^2L(r)^2$.*

PROOF. We only prove the result for **RUPRE**. A similar argument applies to the other algorithms.

$L(b_i)$ is dominated by $L(|A(x)|_\infty) + iL(r)$ so the cost of multiplying $b_i r$ is dominated by $(L(|A(x)|_\infty) + iL(r))L(r)$. Therefore the total time is dominated by

$$\sum_{i=0}^m (L(|A(x)|_\infty) + iL(r))L(r)$$

which is dominated by $mL(|A(x)|_\infty)L(r) + m^2L(r)^2$ ■

Further improvement can be obtained by restricting r to be a binary rational number. In this case $f = 2^k$ and **ITRUNC** can be used to perform the multiplication $a_{m-i}f^i$. The algorithm **IUPBREI**($A(x), r$) assumes $r = e/2^k$ and computes $2^{km} A(e/2^k)$ using the version of Horner's method used by **IUPREI**.

Since f is a power of 2, $A(e/2^k) = 2^{mk-h} A(e/2^k)/2^{mk-h}$, where 2^h is the largest power of two dividing $2^{mk} A(e/2^k)$ and 2^k . The algorithm **IUPBRE** performs this reduction after computing $2^{km} A(e/2^k)$.

Theorem 38 (Computing Time of IUPBREI and IUPBRE) *Let $A(x)$ be an integral polynomial of degree m , and let $r = e/2^k$ be a binary rational number. The algorithm **IUPBREI** computes $2^{km} A(e/2^k)$ and the algorithm **IUPBRE** computes $A(r)$ in time dominated by $mL(|A(x)|_\infty)L(e) + m^2(L(e)^2 + L(e)k)$.*

PROOF. The time for both algorithms is dominated by the cost of computing $2^{km} A(e/2^k)$. Since $L(\bar{b}_i) \preceq L(|A(x)|_\infty) + m(L(e) + k)$, the cost of multiplying $\bar{b}_i e$ is dominated by $(L(|A(x)|_\infty) + m(L(e) + k))L(e)$. The cost of multiplying $a_{m-i}2^{ki}$ with **ITRUNC** is dominated by $L(|A(x)|_\infty) + ki$. Therefore the total cost is dominated by $mL(|A(x)|_\infty)L(e) + m^2L(e)^2 + m^2L(e)k + \sum_{i=0}^m ki \preceq mL(|A(x)|_\infty)L(e) + m^2(L(e)^2 + L(e)k)$ ■

Table 3 empirically compares the various algorithms discussed in this section. The computing times were obtained for integral polynomials of degree m with random coefficients between -2^d and 2^d and binary rational numbers $e/2^k$ where e is a random integer between -2^k and 2^k . All timings are in milliseconds and were averaged over 10 iterations.

3.4 Polynomial Root Bound Computation

In this section we present an implementation, due to Collins, of Theorem 9 for calculating a root bound for integral polynomials. A second technique based on Descartes'

Table 3: Empirical Computing Times (in ms) of Polynomial Evaluation

m	d	k	RUPRE	IUPREI	IUPRE1	IUPRE	IUPBREI	IUPBRE
10	10	10	3.3	1.7	3.3	1.7	1.6	0.0
10	10	155	141.7	26.7	178.4	50.0	20.0	21.7
10	10	300	378.3	75.0	593.5	128.3	63.3	65.1
10	155	10	6.6	1.7	6.7	1.6	3.4	1.6
10	155	155	171.6	46.7	203.3	70.0	23.3	28.4
10	155	300	431.6	113.3	624.9	143.4	75.0	78.3
10	300	10	10.0	5.0	6.7	5.0	3.3	3.4
10	300	155	190.0	60.0	211.6	71.7	28.3	30.0
10	300	300	474.9	143.3	676.8	175.0	83.3	81.7
20	10	10	10.0	5.0	10.0	5.0	1.6	3.4
20	10	155	463.3	103.4	618.2	141.7	85.1	85.0
20	10	300	1310.2	325.0	2243.3	428.2	280.0	285.0
20	155	10	20.0	8.3	15.0	8.4	4.9	6.7
20	155	155	558.3	171.7	728.2	190.0	98.3	96.7
20	155	300	1459.8	431.6	2368.2	535.1	286.6	293.4
20	300	10	20.0	8.3	16.7	8.3	6.7	8.3
20	300	155	636.8	231.6	790.0	271.7	106.7	108.4
20	300	300	1543.5	531.6	2371.7	586.7	295.0	298.4
30	10	10	21.6	8.3	16.7	10.0	3.3	6.6
30	10	155	981.4	233.4	1413.4	263.3	190.0	193.4
30	10	300	2758.6	706.7	4904.7	845.0	620.1	625.0
30	155	10	33.4	16.6	26.6	16.7	8.3	6.7
30	155	155	1142.9	370.0	1546.5	399.9	206.7	203.4
30	300	10	38.3	16.7	30.0	15.0	11.7	13.3
30	300	155	1230.2	473.3	1589.9	541.6	210.1	210.0
30	300	300	3309.8	1200.0	5426.6	1286.8	664.9	665.0

rule of signs is also presented. A theoretical and empirical comparison of the two techniques is given.

Theorem 9 can be used to compute a root bound. However, $|a_{m-k}/a_m|^{1/k}$ can not be computed exactly, so

$$\lceil \log_2 \left(\frac{|a_{m-k}|}{|a_m|} \right) / k \rceil$$

is computed instead. This is done as follows. Let $2^h \leq |a_m| < 2^{h+1}$ and $2^{h_1} \leq |a_{m-k}| < 2^{h_1+1}$, then $2^{(h-h_1)-1} \leq |a_{m-k}|/|a_m| < 2^{(h-h_1)+1}$. Let $d = h - h_1 - 1$ and $d = qk + r$ with $0 \leq r < k$. Then

$$q + r/k \leq \log_2 \left(\frac{|a_{m-k}|}{|a_m|} \right) / k < q + \frac{r+2}{k}.$$

If $r < k - 1$ then

$$\lceil \log_2 \left(\frac{|a_{m-k}|}{|a_m|} \right) / k \rceil = q + 1.$$

If $r = k - 1$ then

$$q \leq \log_2 \left(\frac{|a_{m-k}|}{|a_m|} \right) / k < q + 2,$$

and we check if $\log_2(|a_{m-k}|/|a_m|)/k > q + 1$ by checking if $|a_{m-k}|/|a_m| > 2^{k(q+1)}$.

Using this idea, **IUPRB** computes the smallest power of 2 which is greater than or equal to the bound given in Theorem 9. **IUPRB** is listed in Figure 14.

Theorem 39 (Computing Time of IUPRB) *Let $A(x)$ be an integral polynomial of degree m . Then, the computing time of $\mathbf{IUPRB}(A(x))$ is dominated by $mL(|A(x)|_\infty)$.*

PROOF. By Theorem 29, $\log_2(|a_i|)$ can be computed in time dominated by $L(|A(x)|_\infty)$ using **ILOG2** and this is done for each of the $m + 1$ coefficients ■

$$b \leftarrow \mathbf{IUPRB}(A(x))$$

[Integral univariate polynomial root bound. $A(x)$ is an integral polynomial of positive degree. b is a binary rational number which is a root bound for $A(x)$. If $A(x) = \sum_{i=0}^m a_i x^i$, then b is the smallest power of 2 such that $2|a_{n-k}/a_n|^{1/k} \leq b$.]

1. [Initialize.] $h \leftarrow \log_2(|a_m|)$; $L \leftarrow -\infty$.
2. [Compute root bound.]
 for $k = 1, \dots, m$ do {
 $h_1 \leftarrow \log_2(|a_{m-k}|)$; $d \leftarrow (h - h_1) - 1$; **QREM**(d, k, q, r);
 if $r = k - 1$ then
 if $|a_{m-k}| > |a_m|2^{k(q+1)}$ then $l \leftarrow q + 2$
 else $l \leftarrow q + 1$;
 $L \leftarrow \max(L, l)$; }
 $b \leftarrow 2^{L+1}$ ■

Figure 14: **IUPRB** Integral Univariate Polynomial Root Bound

A similar algorithm based on Theorem 10, **IUPPRB**, is used to compute a bound for the positive roots of $A(x)$.

An alternative algorithm can be designed using Theorem 25. The polynomial $A(x)$ is transformed to $A^*(x) = A(2^k x)$, by repeatedly substituting x by $2x$, until $\text{var}(A^*(x+1)) = 0$. If $\text{var}(A^*(x+1)) = 0$, then 2^k is a positive root bound. A negative root bound can be computed by computing a positive root bound for $A(-x)$.

The algorithm **IUPPRBD**, listed in Figure 15, is more costly than **IUPPRB**; however, it can produce a better root bound.

Theorem 40 (Computing Time of IUPPRBD) *Let $A(x)$ be an integral polynomial of degree m . Then the computing time of $\mathbf{IUPPRBD}(A(x))$ is dominated by*

$$b \leftarrow \mathbf{IUPPRBD}(A(x))$$

[Integral univariate polynomial positive root bound, using Descartes' rule of signs. $A(x)$ is an integral polynomial and b is a bound on the positive roots of $A(x)$.]

1. [Initialize and check for positive roots.] $A^*(x) \leftarrow A(x)$; if $\text{var}(A^*(x)) = 0$ then
 $\{ b \leftarrow 0; \text{return } \}$.
2. [Compute root bound using Descartes' rule and repeated scaling.] $b \leftarrow 1$;
 repeat {
 $n \leftarrow \text{var}(A^*(x+1))$;
 if $n \neq 0$ then {
 $b \leftarrow 2b; A^*(x) \leftarrow A^*(2x)$ }
 until $n = 0$ ■

Figure 15: **IUPPRBD** Integral Univariate Polynomial Positive Root Bound Using Descartes' Rule of Signs

$$m^3 L(|A(x)|_\infty)^2.$$

PROOF. Theorem 20 implies that after the i -th iteration, $|A^*(x)|_\infty \leq 2^{mi} L(|A(x)|_\infty)$ and Theorem 35 implies that $A^*(x+1)$ can be computed in time dominated by $m^2(mi + L(|A(x)|_\infty))$. By Theorems 8 and 25, after at most $L(|A(x)|_\infty)$ iterations $A^*(x+1)$ will have zero variations. Therefore, the total time is dominated by $\sum_{i=1}^{L(|A(x)|_\infty)} m^2(mi + L(|A(x)|_\infty)) \preceq m^3 L(|A(x)|_\infty)^2$ ■

Table 4 compares the computing times and root bounds of **IUPRB**, **IUPPRB**, and **IUPPRBD**. A polynomial

$$A(x) = \left\{ \prod_{i=1}^r (a_i x + b_i) \right\} \left\{ \prod_{j=1}^s (c_j^2 x^2 - 2d_j x + d_j^2 + e_j^2) \right\},$$

where a_i, b_i, c_j, d_j , and e_j are random integers between -2^k and 2^k . The ceiling of the base two logarithm of the largest modulus of the roots, M and the the ceiling of

the base two logarithm of the largest absolute value of the real roots, B are given in Table 4. The algorithm **IUPRB** was used to compute a root bound, and **IUPPRB** and **IUPPRBD** were used to compute positive root bounds. The computing time for each algorithm is listed followed by the root bound it produced. The root bound is 2^{B_i} , where $i = 1$ for **IUPRB**, $i = 2$ for **IUPPRB**, and $i = 3$ for **IUPPRBD**. All timings are in milliseconds and were averaged over 10 iterations.

3.5 Polynomial GCD and Greatest Squarefree Divisor Computation

Some real root isolation algorithm require that the input be squarefree. Therefore, before calling such an algorithm, we must compute the greatest squarefree divisor.

Definition 32 (Greatest Squarefree Divisor) *The greatest squarefree divisor of a polynomial $A(x)$, $\text{gsfd}(A(x))$ is a squarefree polynomial that contains the same zeros as $A(x)$.*

If $A^{(1)}(x)$ is the derivative of $A(x)$, the $\text{gcd}(A(x), A^{(1)}(x))$ has the same zeros as $A(x)$ but each occurs with multiplicity one less than in $A(x)$. Therefore, $\text{gsfd}(A(x)) = A(x) / \text{gcd}(A(x), A^{(1)}(x))$. If $\text{gsfd}(A(x)) = B(x) = \sum_{j=0}^n b_j x^j$, then by Theorem 4 $|\text{gsfd}(A(x))|_1 \leq 2^n |b_n/a_m| |A(x)|_2$. However, the size of the coefficients of a divisor of a polynomial are typically not this large.

The algorithm **IPGCDC**, which implements the Brown-Collins modular polynomial gcd algorithm, is used for computing polynomial gcd's and cofactors.

IPGCDC(1, $A(x)$, $A^{(1)}(x)$; $C(x)$, $\overline{A}(x)$, $\overline{A^{(1)}}(x)$) computes $\text{gsfd}(A(x)) = \overline{A}(x)$. The

Table 4: Empirical Comparison of Root Bound Computation

r	s	k	$L_2(A(x) _\infty)$	B	M	IUPRB	B_1	IUPPRB	B_2	IUPPRBD	B_3
5	0	5	22	3	3	0.16	4	0.34	4	2.66	3
5	0	10	49	2	2	0.17	3	0.17	3	1.33	1
5	0	15	73	3	3	0.33	4	0.34	4	3.16	2
5	5	5	65	1	4	0.50	6	0.33	4	6.50	1
5	5	10	149	4	4	0.66	5	0.50	5	26.84	4
5	5	15	216	5	7	0.84	8	0.66	7	37.84	5
10	0	5	47	5	5	0.33	6	0.50	6	14.50	5
10	0	10	91	7	7	0.17	8	0.16	8	23.67	7
10	0	15	142	5	5	0.50	6	0.83	5	8.17	2
10	5	5	88	2	3	0.67	5	0.33	4	15.17	2
10	5	10	188	5	7	1.00	8	1.50	7	49.66	5
10	5	15	290	2	3	1.16	5	2.17	5	17.34	1
15	0	5	64	3	3	0.50	5	0.67	5	14.50	3
15	0	10	142	3	3	0.83	5	1.00	4	14.17	2
15	0	15	220	8	8	0.50	9	0.50	9	60.02	8
15	5	5	115	5	5	0.83	6	0.50	4	32.51	3
15	5	10	242	3	6	1.50	7	1.50	7	46.00	3
15	5	15	361	5	5	1.50	7	3.00	5	43.51	2
20	0	5	89	5	5	0.67	6	0.33	6	37.83	5
20	0	10	183	5	5	1.17	7	2.00	7	50.68	5
20	0	15	287	6	6	1.50	6	2.16	6	73.84	6
20	5	5	136	5	5	0.67	6	0.66	6	74.52	5
20	5	10	279	5	5	1.34	6	2.82	6	106.84	5
20	5	15	434	6	6	2.00	6	1.33	6	88.33	3

specification for **IPGCD** is listed in Figure ??.

$$\mathbf{IPGCD}(r, A(x), B(x); C(x), \overline{A}(x), \overline{B}(x))$$

[Integral polynomial greatest common divisor and cofactors. $A(x)$ and $B(x)$ are integral polynomials in r variables, $r \geq 0$. $C(x) = \gcd(A(x), B(x))$. If $C(x)$ is non-zero then $\overline{A}(x) = A(x)/C(x)$ and $\overline{B}(x) = B(x)/C(x)$. Otherwise $\overline{A}(x) = 0$ and $\overline{B}(x) = 0$.]

Figure 16: **IPGCD** Integral Polynomial GCD and Cofactor

Theorem 41 (Computing Time of **IPGCD)** *Assume $A(x)$ and $B(x)$ be integral polynomials of degree less than m with $|A(x)|_\infty, |B(x)|_\infty \leq d$. Then the computing time of **IPGCD** is dominated by $m^3L(d)^2$. If $|C(x)|_\infty$, $|\overline{A}(x)|_\infty$, and $|\overline{B}(x)|_\infty$ are also less than d and a negligible number of unlucky primes are encountered, the computing time is dominated by $m^2L(d) + mL(d)^2$. If the inputs are relatively prime and this is detected with a constant number of primes, the computing time is dominated by $m^2 + mL(d)$.*

PROOF. [5] ■

Table 5 lists some empirical computing times for **IPGCD** on randomly generated, dense input polynomials. Three polynomials, $A_1(x)$, $A_2(x)$, and $C(x)$, with k bit coefficients, are generated, and $\gcd(A_1(x)C(x), A_2(x), C(x))$ is computed.

Table 5: Computing Times (in ms) of IPGCDC

$\deg(A_i(x)C(x))$	$\deg(C(x))$	k	IPGCDC
10	0	10	17
10	0	155	84
10	0	300	200
10	0	445	367
10	10	10	0
10	10	155	150
10	10	300	399
10	10	445	767
20	0	10	17
20	0	155	150
20	0	300	366
20	0	445	634
20	10	10	50
20	10	155	732
20	10	300	1800
20	10	445	3333
30	0	10	33
30	0	155	217
30	0	300	500
30	0	445	900
30	10	10	100
30	10	155	1283
30	10	300	3151
30	10	445	5666

CHAPTER IV

Real Root Isolation of Integral Polynomials

In this chapter we derive and analyze three algorithms for isolating the real roots of an integral polynomial. The first algorithm is based on Sturm sequences, the second algorithm is based on Descartes' rule of signs and a series of polynomial transformations, and the third algorithm is based on Rolle's theorem and the derivative sequence. Several variations are discussed for the different algorithms. Maximum computing time bounds are derived for each algorithm and careful empirical comparisons are performed. Besides the computing times, information that characterizes the amount of work that each of the algorithms must perform is recorded. This information helps compare the algorithms independently of a particular implementation. Furthermore, from this data, we are able to conjecture average computing times for the different algorithms.

4.1 Sturm Sequence Based Algorithm

Sturm's theorem suggests an algorithm for isolating the real roots of a polynomial. First a Sturm sequence is computed (this can easily be done since any negative PRS for a polynomial and its derivative is a Sturm sequence). After computing a Sturm sequence Sturm's theorem is applied to an initial interval known to contain all of the

real roots. If no roots are detected the algorithm is done. If a single root is detected, an isolating interval is obtained. If more than one root is in the interval, the interval is bisected and the algorithm is recursively applied to the left and right subintervals. Algorithms based on this approach were first implemented and analyzed by Heindel in his thesis [23]. In this section we review Heindel's results. We also discuss several variations to this algorithm including an improved modular approach to computing Sturm sequences. This approach can be used to determine how many real roots a polynomial has in time bounded by $m^3L(d)^2$, where m is the degree and d is a bound on the coefficients.

4.1.1 Computing Sturm Sequences

If $A(x)$ is squarefree, Theorem 22 implies that any negative PRS for $A(x)$ and $A'(x)$ is a Sturm sequence. The following lemma shows how to convert any PRS into a negative PRS.

Lemma 9 *Let $\{A_1(x), A_2(x), \dots, A_r(x), A_{r+1}(x) = 0\}$ be a PRS defined by $e_i A_i(x) = Q_i(x)A_{i+1}(x) + f_i A_{i+2}(x)$. If $\bar{A}_i(x) = s_i A_i(x)$, where $s_1 = 1$, $s_2 = 1$, and $s_{i+2} = -\text{sign}(e_i)\text{sign}(f_i)\text{sign}(s_i)$ for $i = 1, \dots, r-1$, then $\{\bar{A}_1(x), \bar{A}_1(x), \dots, \bar{A}_r(x), \bar{A}_{r+1}(x) = 0\}$ is a negative PRS for $A_1(x)$ and $A_2(x)$.*

PROOF. Since $e_i s_i A_i(x) = s_i Q_i(x) A_{i+1}(x) + s_i f_i A_{i+2}(x)$, $\bar{e}_i \bar{A}_i(x) = \bar{Q}_i(x) \bar{A}_{i+1}(x) + \bar{f}_i \bar{A}_{i+2}(x)$, where $\bar{e}_i = e_i$, $\bar{Q}_i(x) = s_i s_{i+1} Q_i(x)$ and $\bar{f}_i = s_i s_{i+2} f_i$. Moreover, $\bar{e}_i \bar{f}_i < 0$ since $\text{sign}(\bar{e}_i \bar{f}_i) = -s_{i+2}^2$ ■

For special PRS's $\text{sign}(e_i)$ and $\text{sign}(f_i)$ can be computed without actually computing e_i and f_i . This is the case for the subresultant PRS, where e_i and f_i are products of powers of leading coefficients of the polynomials in the PRS.

All PRS algorithms we will use rely on the computation of the pseudo-remainder. The algorithm **IPPSQR**, listed in Figure 17, computes the pseudo-remainder and pseudo-quotient, of two integral polynomials, while **IPPSR** computes only the pseudo-remainder.

IPPSQR($A, B; Q, R$)

[Integral polynomial pseudo-remainder and quotient. The inputs $A(x)$ and $B(x)$ are integral polynomials with $m = \deg(A(x))$ and $n = \deg(B(x))$. The outputs $Q(x) = \text{pquot}(A(x), B(x))$ and $R(x) = \text{prem}(A(x), B(x))$ are integral polynomials.]

1. $A_0(x) \leftarrow A(x)$;
 for $i = 1, \dots, m - n + 1$ do {
 $A_i(x) \leftarrow b_n A_{i-1}(x) - \text{ldcf}(A_{i-1}(x))B(x)$;
 $Q_i(x) \leftarrow \text{ldcf}(A_{i-1}(x))x^{m-n+1-i} + b_n Q_{i-1}(x)$ };
 $Q(x) \leftarrow Q_{m-n+1}(x)$; $R(x) \leftarrow A_{m-n+1}(x)$ ■

Figure 17: **IPPSQR** Integral Polynomial Pseudo-Remainder and Quotient

It is easy to verify that $Q_{m-n+1}(x)$ is the pseudo-quotient and $A_{m-n+1}(x)$ is the pseudo-remainder.

Theorem 42 (Pseudo-remainder and quotient Coefficient bound.)

Let $m = \deg(A(x))$, $n = \deg(B(x))$, $d = |A(x)|_\infty$, and $e = |B(x)|_\infty$. Then $|\text{prem}(A(x), B(x))|_\infty \leq (2e)^{m-n+1}d$ and $|\text{pquot}(A(x), B(x))|_\infty \leq (2e)^{m-n}d$.

PROOF. We begin by computing a bound for $\text{prem}(A(x), B(x))$. The proof inductively shows that $|A_i(x)|_\infty \leq (2e)^i d$. The theorem is clearly true for $A_0(x) = A(x)$. Assume

that the theorem is true for $A_i(x)$.

$$|A_{i+1}(x)|_\infty \leq |b_n| |A_i(x)|_\infty + |\text{ldcf}(A_i(x))| |B(x)|_\infty \leq 2|B(x)|_\infty |A_i(x)|_\infty,$$

which by induction is less than or equal to $(2e)^{i+1}d$. The bound for $\text{prem}(A(x), B(x))$ is obtained by substituting $m - n + 1$ for i .

A bound on $\text{pquot}(A(x), B(x)) = \sum_{i=0}^{m-n} \text{ldcf}(A_i(x))(b_n x)^{m-n-i}$ follows from the bound for $|A_i(x)|_\infty$.

$$|\text{ldcf}(A_i(x))(b_n x)^{m-n-i}| \leq (2e)^i d e^{m-n-i} \leq (2e)^{m-n} d.$$

Therefore, $|\text{pquot}(A(x), B(x))|_\infty \leq (2e)^{m-n} d$ ■

Theorem 43 (Computing time of IPPSQR) *Let $m = \deg(A(x))$, $n = \deg(B(x))$, $d = |A(x)|_\infty$, and $e = |B(x)|_\infty$. Then the computing time of **IPPSQR** is dominated by $m(m-n)^2 L(e)^2 + m(m-n)L(e)L(d)$.*

PROOF. The time for step i in **IPPSQR** is dominated by the time to compute $A_i(x) = b_n A_{i-1}(x) - \text{ldcf}(A_{i-1}(x))B(x)$, $(m-i)L(e)L(|A_i(x)|_\infty)$, which by Theorem 42 is dominated by $(m-i)L(e)\{iL(e) + L(d)\}$. Therefore, the time for **IPPSQR** is dominated by

$$\sum_{i=0}^{m-n} (m-i)L(e)\{iL(e) + L(d)\} \preceq m(m-n)^2 L(e)^2 + m(m-n)L(e)L(d) \blacksquare$$

The algorithm **IPPNPRS**, listed in Figure 18, uses the **IPPSR** to compute the negative primitive PRS.

$$L \leftarrow \mathbf{IPNPRS}(A(x), B(x))$$

[Integral polynomial primitive negative Polynomial Remainder Sequence. The inputs $A(x)$ and $B(x)$ are integral polynomials with $\deg(A(x)) > 0$, $\deg(A(x)) \geq \deg(B(x))$. The output $L = (A_1(x), A_2(x), A_3(x), \dots, A_r(x))$, where $A_i(x)$ is an integral polynomial and L is the negative primitive PRS for $A_1(x) = A(x)$ and $A_2(x) = B(x)$.]

1. [Initialize.] $A_1(x) \leftarrow A(x)$; $A_2(x) \leftarrow B(x)$; $n_1 \leftarrow \deg(A_1(x))$; $n_2 \leftarrow \deg(A_2(x))$; $L \leftarrow (A_2, A_1)$.
2. [Compute pseudo-remainder.] $A_3(x) \leftarrow \mathbf{IPPSR}(A_1(x), A_2(x))$; if $A_3(x) = 0$ then $\{ L \leftarrow \text{inverse}(L)$; return $\}$.
3. [Make primitive and negative and update.] $A_3(x) := \text{pp}(A_3(x))$; if $\text{ldcf}(A_2(x)) > 0 \vee \text{even}(n_1 - n_2 + 1)$ then $A_3(x) \leftarrow -A_3(x)$; $L \leftarrow (A_3(x), L)$; $A_1(x) \leftarrow A_2(x)$; $n_1 \leftarrow n_2$; $A_2(x) \leftarrow A_3(x)$; $n_2 \leftarrow n_3$; goto 2 ■

Figure 18: **IPNPRS** Integral Polynomial Primitive Negative PRS

Lemma 10 (Primitive PRS Coefficient Bound) *Let $S = \{A_1(x) = A(x), A_2(x) = B(x), \dots, A_r(x), A_{r+1}(x) = 0\}$ be the negative primitive PRS for integral polynomials $A(x)$ and $B(x)$. If $\{m = m_1, m_2, \dots, m_r\}$ is the degree sequence, then $|A_i(x)|_\infty \leq |A(x)|_\infty^{m_1 - m_i} |B(x)|_\infty^{m_1 - m_i}$.*

PROOF. Since $A_i(x)$ is primitive, Theorem 17 implies that $|A_i(x)|_\infty \leq |S_{m_i}(A(x), B(x))|_\infty$ which by Theorem 16 is less than or equal to $|A(x)|_\infty^{m_1 - m_i} |B(x)|_\infty^{m_1 - m_i}$ ■

Theorem 44 (Computing time of IPNPRS) *Let $S = \{A_1(x) = A(x), A_2(x) = B(x), \dots, A_r(x), A_{r+1}(x) = 0\}$ be the negative primitive PRS of A and B . Let $\{m = m_1, m_2, \dots, m_{r+1}\}$ be the corresponding degree sequence and let $|A(x)|_1 \leq d$ and $|B(x)|_1 \leq d$. Then the computing time of $\mathbf{IPNPRS}(A(x), B(x))$ is dominated*

by $m^5L(d)^2$. If S is normal then the computing time of $\mathbf{IPNPRS}(A(x), B(x))$ is dominated $m^4L(d)^2$.

PROOF. Lemma 10 shows that $|A_i(x)|_1 \preceq d^m$. This bound combined with Theorem 43 can be used to obtain a bound on the time required to compute $A_{i+2}(x) = \text{pp}(\text{prem}(A_i(x), A_{i+1}(x)))$.

The time to compute $\text{prem}(A_i(x), A_{i+1}(x))$ is dominated by $m_i\delta_i^2L(d^{m-m_i})^2$ which is dominated by $m^3L(d)\delta_i^2$. Therefore the time to compute all of the pseudo-remainders is dominated by

$$m^3L(d)^2 \sum_{i=3}^r \delta_i^2 \preceq m^3L(d)^2 \left\{ \sum_{i=3}^r \delta_i \right\}^2$$

If S is not normal, $\sum_{i=3}^r \delta_i = m$, so the computing time is dominated by $m^5L(d)^2$.

If S is normal, $\delta_i = 1$ and $\sum_{i=3}^r \delta_i^2 \leq m$ and the computing time is dominated by $m^4L(d)^2$ ■

The subresultant PRS can be used instead of the primitive PRS to obtain an alternative algorithm for computing a Sturm sequence. The algorithm $\mathbf{IPNSPRS}$ (Integral Polynomial Negative PRS) computes the negative subresultant PRS using the recurrence relations in Section 2.3 and Lemma 9 to correct the signs. $\mathbf{IPNSPRS}$ has the same maximum computing time as $\mathbf{IPNPRS}$; however, for typical polynomials it is faster. Another advantage of the subresultant PRS is that it lends itself to modular computation.

The following two theorems, from Rubald [42], are needed to derive a modular algorithm for computing the subresultant PRS. A modular algorithm for computing Sturm sequences was first proposed and implemented by Heindel; however, we present

a modified algorithm which does not need to consider unlucky primes.

Theorem 45 *Assume $A(x)$ and $B(x)$ are polynomials with coefficients in a commutative ring R and that ϕ is a homomorphism from R to $\overline{R}$. If $\deg(\phi(A(x))) = \deg(A(x))$ and $\deg(\phi(B(x))) = \deg(B(x))$, then*

$$\phi(S_k(A(x), B(x))) = S_k(\phi(A(x)), \phi(B(x))).$$

PROOF. The proof follows from the definition of S_k [30] ■

Theorem 46 *Let $\{A_1(x), A_2(x), \dots, A_r(x), A_{r+1}(x) = 0\}$ be a PRS with degree sequence $\{m_1, m_2, \dots, m_r\}$ for $A_1(x)$ and $A_2(x)$. Let $\overline{A}_1(x) = \phi(A_1(x))$ and $\overline{A}_2(x) = \phi(A_2(x))$ and $\{\overline{A}_1(x), \overline{A}_2(x), \dots, \overline{A}_r(x), \overline{A}_{r+1}(x) = 0\}$ be a PRS for $\overline{A}_1(x)$ and $\overline{A}_2(x)$. Then the degree sequence $\{\overline{m}_1, \overline{m}_2, \dots, \overline{m}_r\}$ is a subsequence of $\{m_1, m_2, \dots, m_r\}$.*

PROOF. By Theorem 17 $\overline{A}_i(x)$ is similar to $S_{\overline{m}_i}(\overline{A}_1(x), \overline{A}_2(x))$, which by Theorem 45 is equal to $\phi(S_{\overline{m}_i}(A_1(x), A_2(x)))$. By Theorem 17 $\overline{m}_i$ must be equal to m_j for some j [30] ■

If the degree sequence is a proper subsequence, the homomorphism is called unlucky. It is possible to show that only a finite number of unlucky mappings exist. Unlucky mappings can pose a problem since it is impossible to determine the degree sequence of $A_1(x)$ and $A_2(x)$ from the degree sequence of the image. However, unlucky primes do not pose a problem if the subresultant PRS of the first kind is used, since a polynomial $\overline{A}_k(x)$ of degree $\overline{m}_k$ is equal to $S_{\overline{m}_k}(\overline{A}(x), \overline{B}(x))$ and by Lemma 45 and Lemma 46 is equal to $\phi(S_{m_j}(A(x), B(x)))$ for some j . Therefore $\overline{A}_k(x)$ can be combined with other image polynomials of the same degree.

These two theorems lead to the algorithm **IPNSPRSM**, listed in Figure 19, for computing the negative subresultant PRS for integral polynomials. The subresultant PRS of the first kind is computed modulo p_i (a word-sized prime), using the recurrence relation in Section 2.3, for enough primes to recover the integer subresultant PRS using the Chinese Remainder Theorem. The number of primes is determined by Subresultant coefficient bound in Theorem 16.

$$S \leftarrow \mathbf{IPNSPRSM}(A(x), B(x))$$

[Integral polynomial negative subresultant PRS modular algorithm. $A(x)$ and $B(x)$ are integral polynomials. S is the subresultant PRS for $A(x)$ and $B(x)$.]

1. [Initialize.] $P \leftarrow 1$.
2. [Compute modular images.] Get next prime p_i ; $\bar{A} \leftarrow \phi_{p_i}(A(x))$; $\bar{B} \leftarrow \phi_{p_i}(B(x))$; $\bar{S} \leftarrow \mathbf{SubresultantPRS}(\bar{A}(x), \bar{B}(x))$.
3. [Chinese Remainder Theorem] Use CRT to lift Lift $\bar{S}$ to $Z/(P)$
4. [Coefficient bound.] if $P < \mathbf{Subresultantcoefficientbound}$ then goto 1.
5. [Correct signs.] Use Lemma 9 to convert to a negative PRS ■

Figure 19: **IPNSPRSM** Integral Polynomial Negative Subresultant PRS, Modular Algorithm

Theorem 47 (Computing Time of IPNSPRSM) *Assume $\deg(A(x)) = m$, $\deg(B(x)) = n \leq m$, and $|A(x)|_1 \leq d$ and $|B(x)|_1 \leq d$. Then the computing time of **IPNSPRSM** is dominated by $m^4 L(d)^2$.*

PROOF. Each modular subresultant PRS requires time dominated by m^2 and the time to apply the CRT with t single precision primes is dominated by t^2 . By Theorem 16

$mL(d)$ primes are needed (it is assumed that there are enough single precision primes). Therefore the time for the modular subresultant computation is dominated by $m^3L(d)$ and the time for the applications of the CRT needed to lift all of the coefficients in the PRS is dominated by $m^2(mL(d))^2 = m^4L(d)^2$ ■

4.1.2 Isolating the Real Roots of a Polynomial Using Sturm Sequences

Once a Sturm sequence has been computed, Sturm's theorem can be used to compute isolating intervals for $A(x)$. A positive root bound b_1 is computed and $(0, b_1)$ serves as an initial interval. The Sturm sequence is evaluated at the endpoints of this interval and Sturm's Theorem is used to determine the number of real roots in the interval. If there are no roots in the interval the algorithm stops with no isolating intervals. If there is one root, $(0, b_1]$ is an isolating interval. If there is more than one root, the interval is bisected and the algorithm is called recursively on the left and right subintervals. This process is continued until all of the positive real roots have been isolated. After the positive roots have been isolated the same process is used to isolate the negative roots starting with the interval $(-b_2, 0]$, where $-b_2$ is a negative root bound. The algorithms **IPRIST** and **IPIISS**, listed in Figures 20 and 21, implement this idea.

Algorithms **IPRIST** and **IPIISS** use the subalgorithm **IPLEV** to evaluate a Sturm sequence at a binary rational number. **IPLEV** uses the subalgorithm **IUPBREI** to evaluate $\text{sign}(A_i(x))$ and returns the list $(\text{sign}(A_1(a)), \dots, \text{sign}(A_r(a)))$.

Theorem 48 (Time to Evaluate a Sturm Sequence with IPLEV) *Assume*

$$L \leftarrow \mathbf{IPRIST}(A(x))$$

[Integral polynomial real root isolation using Sturm sequences. The input $A(x)$ is a squarefree integral polynomial. The output $L = (I_1, \dots, I_r)$ is a list of standard isolating intervals for all of the real roots of $A(x)$.]

1. [Compute Sturm sequence.] $\overline{A}(x) \leftarrow \text{pp}(A(x)); \overline{A}'(x) \leftarrow \text{pp}(A'(x));$
 $S \leftarrow \mathbf{NegativePRS}(\overline{A}, \overline{A}')$.
2. [Isolate positive roots.] $b_2 \leftarrow \mathbf{IUPPRB}(\overline{A}); v_2 \leftarrow \text{var}(\mathbf{IPLEV}(S, b_2));$
 $v_0 \leftarrow \text{var}(\mathbf{IPLEV}(S, 0)); L_2 \leftarrow \mathbf{IPIISS}(S, 0, b_2, v_0, v_2)$.
3. [Isolate non-positive roots.] $b_1 \leftarrow \mathbf{IUPPRB}(\overline{A}(-x));$
 $v_1 \leftarrow \text{var}(\mathbf{IPLEV}(S, -b_1)); L_2 \leftarrow \mathbf{IPIISS}(S, -b_1, 0, v_1, v_0)$.
4. [Combine intervals.] $L \leftarrow \text{concat}(L_1, L_2)$ ■

Figure 20: **IPRIST** Integral Polynomial Real Root Isolation Using Sturm Sequences

$$L \leftarrow \mathbf{IPIISS}(S, a, b, v_1, v_2)$$

[Integral polynomial isolating interval search using Sturm sequence. The inputs are S, a, b, v_1, v_2 . $S = (A_1(x), A_2(x), \dots, A_r(x))$ is a list of integral polynomials. S is a Sturm sequence for $A_1(x)$. $a < b$ are binary rational numbers. $v_1 = \text{var}(S, a)$ and $v_2 = \text{var}(S, b)$. The output $L = (I_1, \dots, I_r)$ is a list of standard isolating intervals for the real roots of $A_1(x)$ in the interval $(a, b]$.]

1. [Base Case.] if $v_1 - v_2 = 0$ then { $L \leftarrow ()$; return }
 else if $v_1 - v_2 = 1$ then { $L \leftarrow ((a, b))$; return }.
2. [Evaluate Sturm Sequence at midpoint.] $c \leftarrow (a + b)/2$; ;
 $v \leftarrow \text{var}(\mathbf{IPLEV}(S, c));$.
3. [Check left subinterval $(a, c]$.] $L_1 \leftarrow \mathbf{IPIISS}(S, a, c, v_1, v)$.
4. [Check right subinterval $(c, b]$.] $L_2 \leftarrow \mathbf{IPIISS}(S, c, b, v, v_2)$.
5. [Combine.] $L \leftarrow \text{concat}(L_1, L_2)$ ■

Figure 21: **IPIISS** Integral Polynomial Isolating Interval Search Using Sturm Sequences

$$L \leftarrow \text{IPLEV}(S, a)$$

[Integral polynomial list evaluation of signs. The inputs are $S = (A_1(x), A_2(x), \dots, A_i(x))$ a list of integral polynomials and a a binary rational number. The output $L = (s_1, \dots, s_i)$ where $s_i = \text{sign}(A_i(a))$.]

Figure 22: **IPLEV** Integral Polynomial List Evaluation of Signs

$\deg(A(x)) = m$ and $|A(x)|_\infty \leq d$. Let $S = \{A_1(x), \dots, A_r(x)\}$ be a primitive Sturm sequence for $A(x)$, and let a be a binary rational number. Then the time to compute $\text{IPLEV}(S, a)$ is dominated by $m^3L(d)L(a) + m^3L(a)^2$.

PROOF. By Lemma 10, $|A_i(x)|_\infty \preceq d^m$. Therefore, Theorem 38 implies that the time to compute $A_i(a)$ is dominated by $m^2L(d) + m^2L(a)$. Since there are at most $m + 1$ polynomials in S , the theorem is proved ■

4.1.3 An Example

Let

$$\begin{aligned} A(x) = & 96040x^7 + 105644x^6 - 4373838x^5 - 16035215x^4 - 33253965x^3 - \\ & 44130844x^2 - 30498902x - 7859280 \end{aligned}$$

$A(x)$ has three real roots $(-3/5, -11/2, 8)$ and two pairs of complex conjugate roots $-3/7 \pm 11/7i$ and $-15/14 \pm 1/7i$. The primitive Sturm sequence for $A(x)$,

$$(A_1(x), A_2(x), A_3(x), A_4(x), A_5(x), A_6(x), A_7(x), A_8(x)),$$

is normal and the maximum number of bits of any coefficient in the Sturm sequence is 123 and the average number of bits in the max norms of the polynomials is 60.25.

$$A_1(x) = 96040x^7 + 105644x^6 - 4373838x^5 - 16035215x^4 - 33253965x^3 - \\ 44130844x^2 - 30498902x - 7859280$$

$$A_2(x) = 672280x^6 + 633864x^5 - 21869190x^4 - 64140860x^3 - 99761895x^2 - \\ 88261688x - 30498902$$

$$A_3(x) = 619309824x^5 + 3126834060x^4 + 8605560740x^3 + 14348414555x^2 + \\ 11838660272x + 3515559278$$

$$A_4(x) = 117320275880538220x^4 + 280906311058702980x^3 + \\ 330482236920261267x^2 + 266988056657030832x + \\ 100717446263155358$$

$$A_5(x) = -661234997163947417425197242x^3 - \\ 1878322066603971479485879349x^2 - \\ 1710496467043066818436699790x - 475732559875035760841419988$$

$$A_6(x) = -2997141300636152152710544545267294348x^2 - \\ 5424173021007724749607402606183267133x - \\ 2360335365604312633921900599861947267$$

$$A_7(x) = -1196374706714831516857287317056846327x - \\ 1666292306379710595560216516842659061$$

$$A_8(x) = 1$$

Using this Sturm sequence it is easy to verify that $A(x)$ has three real roots (one positive and two negative). The number of real roots is equal to $V_{-\infty} - V_{\infty}$ which can be computed with the signs of the leading coefficients, and the number of positive roots is equal to $V_0 - V_{\infty}$ which can be computed with the signs of the leading and trailing coefficients.

Even this simple example shows the major disadvantage of an algorithm based on Sturm sequences. Namely, the sizes of the coefficients of the polynomials in a Sturm sequence can be very large, even for the primitive Sturm sequence.

A trace of the algorithm **IPISS** can be represented by a binary tree, where each node in the tree corresponds to a recursive call of the algorithm. Associated with each node is the interval for which that call to **IPISS** is isolating the roots of.

Information about the tree, such as the height, width, and number of nodes, can be used to characterize the behavior of the algorithm. The height, H , of the tree indicates the smallest interval length that is required and the maximum number of bisections required to obtain any interval. The number of interior nodes, I , is equal to the total number of bisections and hence Sturm sequence evaluations. The number of leaf nodes, L , is equal to the number of real roots plus the number of discarded intervals. Since for each recursive call two leaf nodes are added, the trees corresponding to **IPISS** satisfy the following relation: $L = 2I + 1$. From this it follows that $N = I + L = 3I + 1$. Information such as the height and number of nodes can be used to estimate the computing time of **IPISS**.

Each node in the tree can be labeled by (l, n) where $0 \leq l \leq H$ is the level number

and $0 \leq n < 2^l$ is the node number counting from left to right. The interval associated with node (l, n) in the tree for the positive roots is $(nb/2^l, (n+1)b/2^l)$, where b is a positive root bound. In this example, the positive root bound is 16 and the negative root bound is -16 . Figure 23 shows the binary trees associated with the search for the negative roots. Each node in Figure 23 shows the number of variations in the signs of the Sturm sequence evaluated at the endpoints of the interval. The number of real roots in the corresponding interval is just the difference between these two numbers. The output of **IPRIST** is the list of isolating intervals $L = ((-8, -4], (-4, 0], (0, 16])$.

Figure 23: Search Tree associated with **IPRIST**($S, -16, 0, 5, 3$)

4.1.4 Recurrence Relation Evaluation of Sturm Sequences

Let $\{A_1(x), \dots, A_r(x), A_{r+1}(x) = 0\}$ be a PRS of integral polynomials, with degree sequence $\{m_1, \dots, m_r\}$, defined by $e_i A_i(x) = Q_i(x) A_{i+1}(x) = f_i A_{i+2}(x)$. Schwartz and Sharir [45] suggested using the following recurrence relation to evaluate the PRS at a point $a = e/f$. Let $c_i = A_i(a)$. Then

$$c_i = \frac{Q_i(a)c_{i+1} + f_i c_{i+2}}{e_i} \quad (4.1)$$

$$\begin{aligned}
c_{r-1} &= \frac{Q_{r-1}(a)c_r}{e_{r-1}} \\
c_r &= A_r(a).
\end{aligned}$$

Using this relation, all $A_i(a)$ can be computed $A_r(a)$, from the quotient sequence $Q = \{Q_{r-1}(x), \dots, Q_1(x)\}$, and the similarity coefficients $E = \{e_{r-1}, \dots, e_1\}$ and $F = \{f_{r-2}, \dots, f_1\}$. The algorithm **IPPSQSEQ**($A(x), A'(x); Q, E, F$) computes $A_r(x)$, Q , E , and F using the primitive Sturm sequence. In this case $Q_i(x) = \text{pquot}(A_i(x), A_{i+1}(x))$, $\deg(Q_i(x)) = \delta_i = m_i - m_{i+1}$, $e_i = \text{lcf}(A_{i+1}(x))^{\delta_i}$, and $f_i = \pm \text{cont}(\text{prem}(A_i(x), A_{i+1}(x)))$.

This method seems preferable to directly evaluating the PRS since the sum of the degrees of the quotient sequence is m whereas the sum the degrees of the remainder sequence is m^2 . Moreover, if the PRS is normal then each quotient polynomial $Q_i(x)$ is linear. However, the recurrence method is not the case if the size of a is small compared to $mL(d)$.

The recurrence relation in Equation 4.1 involves rational arithmetic with costly integers gcd operations. This can be remedied by computing $\bar{c}_i = f^{m_i} A_i(e/f)$ instead of c_i .

$$\bar{c}_i = \frac{f^{\delta_i} Q_i(e/f) \bar{c}_{i+1} + f^{\delta_i + \delta_{i+1}} f_i \bar{c}_{i+2}}{e_i}, \quad (4.2)$$

which only involves integer arithmetic. This suffices for computing the sign of $A_i(a)$ since $\text{sign}(c_i) = \text{sign}(\bar{c}_i)$. Algorithm **IPRSEVS**, whose specification is listed in Figure 24, implements this recurrence relation.

$$L \leftarrow \text{IPRSEVS}(Q, E, F, a)$$

[Integral polynomial Recursive Sturm Sequence Evaluation of Signs. The inputs are Q , E , F , and a . $Q = (A_r(x), Q_{r-1}(x), \dots, Q_1(x))$ is a list of integral polynomials, $E = (e_{r-1}, \dots, e_1)$ is a list of integers, and $F = (f_{r-2}, \dots, f_1)$ is a list of integers, where $e_i A_i(x) = Q_i(x) A_{i+1}(x) + f_i A_{i+2}(x)$. a is a binary rational number. The output $L = (s_1, \dots, s_r)$, where $s_i = \text{sign}(A_i(a))$.]

Figure 24: **IPRSEVS** Integral Polynomial Recursive Sturm Sequence Evaluation of Signs

Theorem 49 (Computing time of IPRSEVS.) *Let*

$$S = \{A_1(x), \dots, A_r(x), A_{r+1}(x) = 0\}$$

*be a primitive PRS with degree sequence $\{m = m_1, \dots, m_{r+1}\}$ and let $\delta_i = m_i - m_{i+1}$. Assume $|A_1(x)|_1 \leq d$, $|A_2(x)|_1 \leq d$, and a is a binary rational number. Then computing time of **IPRSEVS** is dominated by $m^3 L(d)^2 + m^3 L(d) L(a) + mL(a)^2$.*

PROOF. $L(\bar{e}_i) \preceq m_i L(a) + L(|A_i|_\infty) \preceq m (L(a) + L(d))$. Since $Q_i(x) = \text{pquot}(A_i(x), A_{i+1}(x))$, Theorem 42 implies that $L(|Q_i(x)|_\infty) \preceq L(|A_i|_\infty |A_{i+1}|_\infty^{\delta_i}) \preceq \delta_i m L(d)$. Using this bound, $L(f^{\delta_i} Q_i(e/f)) \preceq L(|Q_i(x)|_\infty) + \delta_i L(a) \preceq \delta_i (m L(d) + L(a))$.

Since $e_i = \text{ldcf}(A_{i+1}(x))^{\delta_i}$, $L(e_i) \preceq \delta_i m L(d)$. Since $f_i = \pm \text{cont}(\text{prem}(A_i(x), A_{i+1}(x)))$, $L(f_i) \preceq L(|\text{prem}(A_i(x), A_{i+1}(x))|_\infty) \preceq \delta_i m L(d)$.

These bounds can now be used to bound the time required to compute $\bar{e}_i$ using the recurrence relation in Equation 4.2. By Theorem 38, the polynomial evaluation $f^{\delta_i} Q(a)$ is computed in time dominated by $\delta_i^2 L(a)^2 + \delta_i^2 L(|Q_i(x)|_\infty) L(a)$, which is dominated by $\delta_i^2 L(d) L(a) + \delta_i^2 L(a)^2$. The multiplication of this result by $\bar{e}_{i+1}$ requires

$L(f^{\delta_i}Q(a))L(\bar{c}_{i+1})$ operations, which is dominated by $\delta_i m^2 L(d)^2 + \delta_i m^2 L(d)L(a) + \delta_i m L(a)^2$. The multiplication of $f_i f^{m_i - m_{i+2}} \bar{c}_{i+2}$ requires $L(f_i)L(f^{m_i - m_{i+2}} \bar{c}_{i+2})$ operations, which is dominated by $\delta_i m^2 L(d)^2 + \delta_i m^2 L(d)L(a)$. Finally, the division by e_i requires $L(\bar{c}_i)L(e)$ operations, which is dominated by $\delta_i m^2 L(d)^2 + \delta_i m^2 L(d)L(a)$.

Combining the cost of these four operations, $\bar{c}_i$ can be computed in time dominated by $\delta_i (m^2 L(d)^2 + m^2 L(d)L(a) + m L(a)^2)$. Therefore the total time is dominated by

$$(m^2 L(d)^2 + m^2 L(d)L(a) + m L(a)^2) \sum_{i=1}^r \delta_i,$$

which is dominated by $m^3 L(d)^2 + m^3 L(d)L(a) + m^2 L(a)^2$ ■

Table 6 compares the theoretical computing times of **IPRSEVS** and **IPLEV**.

$L(a)$	IPLEV	IPRSEVS
$L(a) \sim 1$	$m^3 L(d)$	$m^3 L(d)^2$
$L(a) \sim L(d)$	$m^3 L(d)^2$	$m^3 L(d)^2$
$L(a) \sim mL(d)$	$m^5 L(d)^2$	$m^4 L(d)^2$

Table 6: Comparison of **IPLEV** and **IPRSEVS**

Table 6 suggests that **IPLEV** performs better in practice where $L(a)$ is likely to be codominant with 1 or $L(d)$, but **IPRSEVS** is better in the worst case.

4.2 An Algorithm Based on Polynomial Transformations and Descartes' Rule of Signs

The coefficient sign variation method, for polynomial real root isolation, is based on two special cases of Descartes' rule of signs and the use of polynomial transformations. Descartes rule (Theorem 24) can be applied to obtain an upper bound for the number

of positive roots of a polynomial $A(x)$. An upper bound can be obtained on the number of roots of $A(x)$ in the interval (a, b) by first mapping the roots of $A(x)$ in the interval (a, b) to the positive roots of $A^*(x)$ and then applying Descartes' rule to $A^*(x)$.

The linear fractional transformation

$$L = \begin{pmatrix} 1/(b-a) & -a/(b-a) \\ 0 & 1 \end{pmatrix} \approx \begin{pmatrix} 1 & -a \\ 0 & b-a \end{pmatrix}$$

maps the circle in the complex plane, whose diameter is the interval (a, b) , to the circle, C , whose diameter is the normalized interval $(0, 1)$. Moreover, L maps the interval (a, b) to the normalized interval $(0, 1)$. Geometrically, L can be viewed as a translation, T_{-a} , which translates the interval (a, b) to $(0, b-a)$, followed by the homothetic transformation, $H_{1/(b-a)}$, which scales the interval $(0, b-a)$ to $(0, 1)$.

The interval $(0, 1)$ can then be mapped to the semi-infinite interval $(0, \infty)$ by mapping the circle C to the right half plane. The dual transformation, $D : z \rightarrow 1/z - 1$ accomplishes this. Geometrically, the dual transformation can be viewed as an inversion, which maps the circle C to the line $\Re(z) = 1$ and the interior of C to the half plane $\Re(z) > 1$, followed by a translation by minus one. Combining these transformations, we obtain a transformation, L^* which maps (a, b) to $(0, \infty)$.

$$L^* = DL = \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & -a \\ 0 & b-a \end{pmatrix} = \begin{pmatrix} -1 & b \\ 1 & -a \end{pmatrix},$$

The positive roots of the transformed polynomial $A^*(x) = L^*(A(x)) = (x + 1)^m A(ax + b)$, correspond to the roots of $A(x)$ in the interval (a, b) . This transformation can be factored into a product of homothetic transformations, translations and inversions: $L^* = T_{-1}RH_{1/(b-a)}T_{-a}$.

For standard intervals, the homothetic transformation becomes a binary homothetic transformation. The transformation mapping the standard interval $(a/2^e, (a+1)/2^e)$, to $(0, \infty)$ is equal to DL , where

$$L = \begin{pmatrix} 2^e & -a \\ 0 & 1 \end{pmatrix}.$$

The transformation $L = H_{2^e} T_{-a/2^e} = T_{-a} H_{2^e}$. The latter factorization is preferable since the required translation is an integer translation.

To determine the number of roots of $A(x)$ in the interval (a, b) , we compute the transformed polynomial $A^*(x) = L^*(A(x))$ and count the number of coefficient sign variations of $A^*(x)$. If $\text{var}(A^*(x)) = 0$, then $A(x)$ does not have any roots in (a, b) , if $\text{var}(A^*(x)) = 1$, then $A(x)$ has a single root in (a, b) , and if $\text{var}(A^*(x)) > 1$ then the number of roots is undetermined. In the last case, $\text{var}(A^*(x)) > 1$, the interval (a, b) must be bisected and the left and right subintervals tested in the same way.

Let $I = (a/2^e, (a+1)/2^e)$ be a standard interval and let $I_1 = (a/2^e, (2a+1)/2^{e+1})$ and $I_2 = ((2a+1)/2^{e+1}, (a+1)/2^e)$ be the left and right subintervals obtained after bisection. Let

$$L = \begin{pmatrix} 2^e & -a \\ 0 & 1 \end{pmatrix}, \quad L_1 = \begin{pmatrix} 2^{e+1} & -2a \\ 0 & 1 \end{pmatrix}, \quad \text{and} \quad L_2 = \begin{pmatrix} 2^{e+1} & -2a-1 \\ 0 & 1 \end{pmatrix},$$

and let $A_1(x) = L_1(A(x))$ and $A_2(x) = L_2(A(x))$. Then $DL(A(x)) = A^*(x)$ is a polynomial whose positive roots correspond to the roots of $A(x)$ in I , $DL_1(A(x)) = A_1^*(x)$ is a polynomial whose positive roots correspond to the roots of $A(x)$ in I_1 , and $DL_2(A(x)) = A_2^*(x)$ is a polynomial whose positive roots correspond to the roots of $A(x)$ in I_2 . Since $L_1 = H_2 L$, $A_1(x) = H_2(L(A(x)))$ is obtained from $L(A(x))$

by applying a binary homothetic transformation, and since $L_2 = T_{-1}L_1$, $A_2(x) = T_{-1}(A_1(x))$ is obtained with a translation by minus one.

These ideas are used by the algorithm **DESCARTES**, listed in Figure 25, to determine the number of real roots of $A(x)$ in the standard interval $I = (a, b]$. The algorithm checks if $A(b) = 0$ by testing if $x|A(x)$.

$$n \leftarrow \mathbf{DESCARTES}(\tilde{A}(x), I)$$

[Compute the number of real roots of $A(x)$ in the interval I using Descartes' rule of signs and polynomial transformations. Let L be the linear fractional transformation that maps I to the normalized interval $(0, 1)$. $\tilde{A}(x) = L(A(x))$ is a squarefree integral polynomial. I is a left-open and right-closed standard interval. n is number of real roots of $A(x)$ in I .]

1. [Base case.] $n \leftarrow 0$;
 if $x|\tilde{A}(x)$ then $\{ n \leftarrow 1; \overline{A}(x) \leftarrow \tilde{A}(x)/x \}$
 else $\{ \overline{A}(x) = \tilde{A}(x) \}$;
 if $\text{var}(D(\overline{A}(x))) = 0$ then $\{ \text{return} \}$
 else if $\text{var}(D(\overline{A}(x))) = 1$ then $\{ n \leftarrow n + 1; \text{return} \}$
 else $\{ A_1(x) = H_2(\overline{A}(x)); A_2(x) = T_{-1}(A_1(x)) \}$.
2. [Recursion.] $n_1 \leftarrow \mathbf{DESCARTES}(A_1(x), I_1)$;
 $n_2 \leftarrow \mathbf{DESCARTES}(A_2(x), I_1)$; $n \leftarrow n_1 + n_2$ ■

Figure 25: Algorithm **DESCARTES**

Termination of this algorithm is guaranteed by Theorem 25 and Theorem 27, which imply that if there is zero or one real roots in the interval (a, b) and the remaining real and complex roots are far enough away, the transformed polynomial $DL(A(x))$ will have zero and one variations respectively. This proves termination since each bisection moves the roots apart by a factor of two.

4.2.1 The Coefficient Sign Variation Method for Real Root Isolation

The ideas used by the algorithm **DESCARTES** can be used to isolate the real roots of a squarefree polynomial. The approach we take isolates the positive and negative roots separately. The algorithm first decides whether $A(0) = 0$ by checking if $x|A(x)$. Then a positive root bound, b_p , is computed and the roots of $A(x)$ in the interval $(0, b_p)$ are isolated. The normalized polynomial $\tilde{A}_p(x) = H_{1/b_p}(A(x))$, whose roots in the normalized interval $(0, 1)$ correspond to the positive roots of $A(x)$, is computed. Since the algorithms we use for computing root bounds return a root bound which is a power of two, a binary homothetic transformation can be used. The subalgorithm **IPRINCS** is then used to isolate the real roots of $\tilde{A}_p(x)$ in the interval $(0, 1)$. **IPRINCS** computes the number of variations of $\text{var}(D(\tilde{A}_p(x)))$. If $\text{var}(D(\tilde{A}_p(x))) = 0$ then there are no positive roots of $\tilde{A}_p(x)$. If $\text{var}(D(\tilde{A}_p(x))) = 1$ there is a single positive root of $\tilde{A}_p(x)$ and $(0, 1)$ is an isolating interval. If $\text{var}(D(\tilde{A}_p(x))) > 1$, the interval $(0, 1)$ is bisected and **IPRINCS** recursively checks the left and right subintervals, using the bisection polynomials $\tilde{A}_1(x) = H_2(\tilde{A}_p(x))$ and $\tilde{A}_2(x) = T_{-1}(\tilde{A}_1(x))$ for the left and right subintervals respectively. After **IPRINCS** has isolated the roots of $\tilde{A}_p(x)$ in the interval $(0, 1)$, isolating intervals for the positive roots of $A(x)$ are obtained by scaling the isolating intervals returned by **IPRINCS**.

After the positive roots have been isolated, the negative roots are isolated by isolating the positive roots of $A_n(x) = A(-x)$. A negative root bound, b_n , is computed and **IPRINCS** is used to isolate the roots of $\tilde{A}_n(x) = H_{1/b_n}(A_n(x))$. The algorithms

IPRICS and **IPRINCS** are listed in Figures 26 and 27.

$$L \leftarrow \mathbf{IPRICS}(A(x))$$

[Integral polynomial real root isolation, coefficient sign variation method. $A(x)$ is a squarefree integral polynomial. $L = (I_1, \dots, I_r)$ is a list of isolating intervals for the real roots of $A(x)$. $I_j = (a_j, b_j)$ is standard open or one-point binary rational interval and $a_1 \leq b_1 \leq \dots \leq a_r \leq b_r$.]

1. [Initialize and check if $A(0) = 0$.]
 if $x|A(x)$ then { $L_0 \leftarrow ((0, 0))$; $A(x) = x/A(x)$; }
 else $L_0 \leftarrow ()$;
 $L_p \leftarrow ()$; $L_n \leftarrow ()$.
2. [Isolate positive roots.]
 $b \leftarrow \mathbf{IUPPRB}(A(x))$; $\tilde{A}_p \leftarrow H_{1/b_p}(A(x))$;
 $L_p \leftarrow \mathbf{IPRINCS}(\tilde{A}_p(x), 0, 1)$;
 Scale intervals in L_p by b_p .
3. [Isolate negative roots.]
 $A_n(x) \leftarrow A(-x)$; $b_n \leftarrow \mathbf{IUPPRB}(A_n(x))$; $\tilde{A}_n(x) \leftarrow H_{1/b_n}(A_n(x))$;
 $L_n \leftarrow \mathbf{IPRINCS}(\tilde{A}_n(x), 0, 1)$;
 Scale intervals in L_n by $-b_n$.
4. [Combine.] $L \leftarrow \text{concat}(L_n, L_0, L_p)$ ■

Figure 26: **IPRICS** Integral Polynomial Root Isolation, Coefficient Sign Variation Method

The algorithm **IPRINCS** is characterized, in the same way as the Sturm sequence algorithm **IPHISS**, by a binary tree, where each node corresponds to recursive call of the algorithm or a subinterval of $(0, 1)$. Corresponding to each node in the tree is a polynomial whose positive roots correspond to the roots of $A(x)$ in the associated interval. The interval $(n/2^l, (n+1)/2^l)$ and the polynomial $T_{-n}H_{2^l}(A(x))$ are associated with the node (l, n) . For each internal node, **IPRINCS** performs one

$$L \leftarrow \text{IPRINCS}(A(x), I)$$

[Integral polynomial real root isolation, normalized coefficient sign variation method. $A(x)$ is a squarefree integral polynomial. $I = (a, b)$ is a standard interval. $L = (I_1, \dots, I_r)$ is a list of isolating intervals for $T(A(x))$ in the interval $(0, 1)$, where T is the linear fractional transformation that maps (a, b) onto $(0, 1)$. $I_j = (a_j, b_j)$ is either a standard open or one-point interval and $a_1 \leq b_1 \leq \dots \leq a_r \leq b_r$.]

1. **[Initialize and check if $A(a) = 0$.]**
 $a \leftarrow \text{LeftEndpoint}(I)$; $b \leftarrow \text{RightEndpoint}(I)$;
 if $x|A(x)$ then $\{ L_0 \leftarrow ((a, a)); A(x) \leftarrow A(x)/x \}$
 else $L_0 \leftarrow ()$.
2. **[Base case.]** $A^*(x) \leftarrow D(A(x))$;
 if $\text{var}(A^*(x)) = 0$ then $\{ L \leftarrow L_0$; return $\}$; if $\text{var}(A^*(x)) = 1$ then $\{$
 $L \leftarrow \text{concat}(L_0, ((a, b)))$; return $\}$.
3. **[Bisect.]** $c \leftarrow (a + b)/2$;
 $A_1(x) \leftarrow H_2(A(x))$; $A_2(x) \leftarrow T_{-1}(A_1(x))$.
4. **[Left recursive call.]**
 $L_1 \leftarrow \text{IPRINCS}(A_1(x), (a, c))$.
5. **[Right recursive call.]**
 $L_2 \leftarrow \text{IPRINCS}(A_2(x), (c, b))$.
6. **[Combine.]** $L \leftarrow \text{concat}(L_1, L_0, L_2)$ ■

Figure 27: **IPRINCS** Integral Polynomial Root Isolation, Normalized Coefficient Sign Variation Method

polynomial inversion, R , one homothetic transformation, H_2 , and two translations, T_{-1} , and for each leaf node **IPRINCS** performs one polynomial inversion and one translation, T_{-1} . Let N equal the number of nodes in the tree, L equal the number of leaf nodes, and I equal the number of internal nodes. Then the number of inversions is equal to $I + L = N$, the number of homothetic transformations is equal to I and the number of translations is equal to $2I + L$.

Figure 28 shows the tree for the negative roots of the example $A(x) = 96040x^7 + 105644x^6 - 4373838x^5 - 16035215x^4 - 33253965x^3 - 44130844x^2 - 30498902x - 7859280$. Each node lists the number of variations obtained for $D(A(x))$ where $A(x)$ is the polynomial corresponding to the node. The corresponding polynomials are listed below. This tree has several more nodes than the one associated with **IPRISS** and illustrates that since Descartes' rule is not exact more bisections may be required than when using Sturm sequences.

$$\begin{aligned}
A_{(0,0)} &= -1611283824640x^7 + 110775762944x^6 + 286643847168x^5 - \\
&\quad 65680240640x^4 + 8513015040x^3 - 706093504x^2 + 30498902x - 491205 \\
A_{(1,0)} &= -12588154880x^7 + 1730871296x^6 + 8957620224x^5 - 4105015040x^4 + \\
&\quad 1064126880x^3 - 176523376x^2 + 15249451x - 491205 \\
A_{(2,0)} &= -196689920x^7 + 54089728x^6 + 559851264x^5 - 513126880x^4 + \\
&\quad 266031720x^3 - 88261688x^2 + 15249451x - 982410 \\
A_{(3,0)} &= -3073280x^7 + 1690304x^6 + 34990704x^5 - 64140860x^4 +
\end{aligned}$$

$$66507930x^3 - 44130844x^2 + 15249451x - 1964820$$

$$A_{(4,0)} = -96040x^7 + 105644x^6 + 4373838x^5 - 16035215x^4 + 33253965x^3 - \\ 44130844x^2 + 30498902x - 7859280$$

$$A_{(4,1)} = -96040x^7 - 566636x^6 + 2990862x^5 + 4057235x^4 + 11602965x^3 + \\ 2725961x^2 - 310977x + 110970$$

$$A_{(3,1)} = -3073280x^7 - 19822656x^6 - 19406352x^5 + 28602420x^4 + \\ 86092810x^3 + 81270506x^2 + 33530497x + 5128585$$

$$A_{(2,1)} = -196689920x^7 - 1322739712x^6 - 3246098688x^5 - 3786671840x^4 - \\ 1990315800x^3 - 89557568x^2 + 331278963x + 96161265$$

$$A_{(1,1)} = -12588154880x^7 - 86386212864x^6 - 245008404480x^5 - \\ 373939265280x^4 - 331747725920x^3 - 170426213776x^2 - \\ 46509232085x - 5102316650$$

4.2.2 The Dual Algorithm

An alternative algorithm, called the dual algorithm can be obtained by directly computing the test polynomials $A_1^*(x)$ and $A_2^*(x)$ from $A^*(x)$ rather applying the dual transformation, D to the bisection polynomial $A_1(x)$ and $A_2(x)$. Let $A_1^*(x) = T_1^*(A^*(x))$ and $A_2^*(x) = T_2^*(A^*(x))$. Since $DH_2(A(x)) = T_1^*D(A(x))$,

$$T_1^* = DH_2D^{-1} = \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}.$$

Since $T_1^* = H_{1/2}T_{-1}$ it can be computed with a translation by minus one followed by a binary homothetic transformation.

Figure 28: Search Tree for **IPRINCS**

Similarly, since $DT_{-1}H_2(A(x)) = T_2^*D(A(x))$,

$$T_2^* = DT_{-1}H_2D^{-1} = \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ -1 & 1 \end{pmatrix}.$$

Since $T_2^* = RT_1^*R$, it can be computed with two inversions, a translation by minus one and a homothetic transformation.

Using these transformations we arrive at the dual algorithms **IPRICSD** and **IPRINCSD**, listed in Figures 29 and 30.

The binary tree associated with **IPRINCSD** is the same as the one associated with **IPRINCS**. The polynomial associated with node (l, n) is obtained from the polynomial associated with the same node in the tree for **IPRINCS** by applying the dual transformation. Therefore, the size of the coefficients of the polynomial in the dual tree are potentially 2^m times as large as the coefficients of the polynomial in the tree for **IPRINCS**. The transformed polynomials of the dual algorithm for the same example used for **IPRINCS** are listed below. However, the number of transformations performed by the dual algorithm is not the same as the number performed by **IPRINCS**. For each internal node, **IPRINCSD** performs two polynomial inversions, two homothetic transformation, $H_{1/2}$, and two translations, T_{-1} , and for each leaf node **IPRINCSD** does not perform any transformations. Therefore, the total number of inversions is equal to $2I$, the total number of homothetic transformations is equal to $2I$ and the total number of translations is equal to $2I$. Thus L fewer translations are needed, I more homothetic transformations are needed, and $L - I = I + 1$ fewer inversions are needed (this follows from $L = 2I + 1$). We end this section by listing the dual polynomials corresponding to the example in Section 4.2.1.

$$L \leftarrow \text{IPRICSD}(A(x))$$

[Integral polynomial real root isolation, coefficient sign variation method, dual algorithm. $A(x)$ is a squarefree integral polynomial. $L = (I_1, \dots, I_r)$ is a list of isolating intervals for the real roots of $A(x)$. $I_j = (a_j, b_j)$ is standard open or one-point binary rational interval and $a_1 \leq b_1 \leq \dots \leq a_r \leq b_r$.]

1. [Initialize and check if $A(0) = 0$.]
 if $x|A(x)$ then { $L_0 \leftarrow ((0,0)); A(x) = x/A(x);$ }
 else $L_0 \leftarrow ()$;
 $L_p \leftarrow (); L_n \leftarrow ()$.
2. [Isolate positive roots.]
 $b \leftarrow \text{IUPPRB}(A(x)); \tilde{A}_p \leftarrow H_{1/b_p}(A(x));$
 $L_p \leftarrow \text{IPRINCSD}(D(\tilde{A}_p(x)), 0, 1);$
 Scale intervals in L_p by b_p .
3. [Isolate negative roots.]
 $A_n \leftarrow A(-x); b_n \leftarrow \text{IUPPRB}(A_n(x)); \tilde{A}_n(x) \leftarrow H_{1/b_n}(A_n(x));$
 $L_n \leftarrow \text{IPRINCSD}(D(\tilde{A}_n(x)), 0, 1);$
 Scale intervals in L_n by $-b_n$.
4. [Combine.] $L \leftarrow \text{concat}(L_n, L_0, L_p)$ ■

Figure 29: **IPRICSD** Integral Polynomial Root Isolation, Coefficient Sign Variation Method, Dual Algorithm

$$L \leftarrow \text{IPRINCSD}(A^*(x), I)$$

[Integral polynomial real root isolation, normalized coefficient sign variation method, dual algorithm. $A^*(x)$ is a squarefree integral polynomial. $I = (a, b)$ is a standard interval. numbers. $L = (I_1, \dots, I_r)$ is a list of isolating intervals for $T(A^*(x))$ in the interval $(0, 1)$, where T is the linear fractional transformation that maps (a, b) onto $(0, \infty)$. $I_j = (a_j, b_j)$ is either a standard open or one-point interval and $a_1 \leq b_1 \leq \dots \leq a_r \leq b_r$.]

1. [Initialize and check if $A(b) = 0$.]
 $a \leftarrow \text{LeftEndpoint}(I)$; $b \leftarrow \text{RightEndpoint}(I)$;
 if $x|A^*(x)$ then $\{ L_0 \leftarrow ((b, b)); A^*(x) \leftarrow A^*(x)/x \}$
 else $L_0 \leftarrow ()$.
2. [Base case.] if $\text{var}(A^*(x)) = 0$ then $\{ L \leftarrow L_0$; return $\}$; if $\text{var}(A^*(x)) = 1$ then
 $\{ L \leftarrow \text{concat}(L_0, ((a, b)))$; return $\}$.
3. [Bisect.] $c \leftarrow (a + b)/2$;
 $A_1^*(x) \leftarrow H_{1/2}T_{-1}(A^*(x))$; $A_2^*(x) \leftarrow RH_{1/2}T_{-1}R(A^*(x))$.
4. [Left recursive call.]
 $L_1 \leftarrow \text{IPRINCSD}(A_1^*(x), a, c)$.
5. [Right recursive call.]
 $L_2 \leftarrow \text{IPRINCSD}(A_2^*(x), c, b)$.
6. [Combine.] $L \leftarrow \text{concat}(L_1, L_0, L_2)$ ■

Figure 30: IPRINCSD Integral Polynomial Root Isolation, Normalized Coefficient Sign Variation Method, Dual Algorithm

$$\begin{aligned}
A_{(0,0)}^* &= -491205x^7 - 33937337x^6 - 899402221x^5 - 12518158265x^4 - \\
&\quad 107420406055x^3 - 542291393203x^2 - 697321612839x + 1360485401125 \\
A_{(1,0)}^* &= -491205x^7 + 11811016x^6 - 95341975x^5 + 393059590x^4 - \\
&\quad 1325944435x^3 + 1480529084x^2 + 10793015535x - 5102316650 \\
A_{(2,0)}^* &= -982410x^7 + 8372581x^6 - 17395592x^5 + 19080695x^4 - 61012210x^3 - \\
&\quad 57844781x^2 + 341849892x + 96161265 \\
A_{(3,0)}^* &= -1964820x^7 + 1495711x^6 + 6104642x^5 + 5826775x^4 - 3197260x^3 - \\
&\quad 12212191x^2 + 2369598x + 5128585 \\
A_{(4,0)}^* &= -7859280x^7 - 24516058x^6 - 26182312x^5 - 4991525x^4 + \\
&\quad 10575445x^3 + 6922193x^2 + 1087767x + 110970 \\
A_{(4,1)}^* &= 110970x^7 + 465813x^6 + 3190469x^5 + 24452065x^4 + 75393115x^3 + \\
&\quad 109705682x^2 + 76539386x + 20514340 \\
A_{(3,1)}^* &= 5128585x^7 + 69430592x^6 + 390153773x^5 + 1174903270x^4 + \\
&\quad 2035789135x^3 + 2006320568x^2 + 1014978747x + 192322530 \\
A_{(2,1)}^* &= 96161265x^7 + 1004407818x^6 + 3917502775x^5 + 5896725080x^4 - \\
&\quad 2652287185x^3 - 20455013678x^2 - 24923201015x - 10204633300 \\
A_{(1,1)}^* &= -5102316650x^7 - 82225448635x^6 - 556630255936x^5 - \\
&\quad 2060098358825x^4 - 4513958031170x^3 - 5866361824525x^2 - \\
&\quad 4192114399284x - 1271707525935
\end{aligned}$$

4.3 An Algorithm Based on Rolle's Theorem and the Derivative Sequence

A major problem with the Sturm sequence algorithm is that the sizes of the coefficients of the polynomials in the Sturm sequence are typically very large, and hence it can be costly to compute and evaluate Sturm sequences. Therefore, alternative methods are sought which do not involve polynomials with such large sized coefficients. One possibility is to use the derivative sequence instead of a Sturm sequence and Fourier's theorem instead of Sturm's theorem. This is promising since the size of the coefficients of the primitive derivative sequence are typically much smaller than those of the primitive Sturm sequence. Thus we can try to replace the use of Sturm Sequences in the algorithm **IPRIST** by the derivative sequence. Even though Fourier's theorem only gives an upper bound on the number of roots in an interval, it gives the exact answer if zero or one variations are obtained. Therefore, this algorithm proceeds exactly as the Sturm sequence algorithm does. If $V_a - V_b = 0$, the interval $(a, b]$ is discarded, if $V_a - V_b = 1$, $(a, b]$ is an isolating interval, and if $V_a - V_b > 1$, the interval $(a, b]$ is bisected.

Unfortunately, this does not lead to an algorithm since it may never terminate. For example, consider $A(x) = x^2 + x + 1$ and the interval $(-1, 0]$. The derivative sequence is $A(x)$, $A^{(1)}(x) = 2x + 1$, $A^{(2)}(x) = 2$. Let

$$V_a = \text{var}(\text{sign}(A(a)), \text{sign}(A^{(1)}(a)), \text{sign}(A^{(2)}(a))).$$

Then $V_{-1} - V_0 = 2$ and I must be bisected. The left subinterval, $(-1, -1/2]$ must also be bisected since $V_{-1} - V_{-1/2} = 2$. Moreover, $V_a - V_{-1/2} = 2$ for all $a < -1/2$, so

the subinterval with $-1/2$ as an endpoint will always be bisected and the algorithm will never determine that there are no roots in the interval $(-1, 0]$. In fact, any time $V_a - V_b$ is greater than the number of roots in $(a, b]$ the algorithm will not terminate since $V_a - V_b = (V_a - V_c) + (V_c - V_b)$ for all $c \in (a, b)$ and at least one of the subintervals will contain more variations than roots.

Despite the failure of this approach, the derivative sequence can still be used to derive a valid algorithm. One possible approach was presented by Collins and Loos in their paper [14]. Their algorithm inductively obtains isolating intervals for a polynomial from isolating intervals for its derivative. The algorithm is based on Rolle's theorem which implies that any interval containing at most one root of the derivative can contain at most two roots of the polynomial. Using this basic idea, the algorithm does a case by case analysis, based on the signs of the polynomial and its derivative at the endpoints of the intervals, in order to find isolating intervals for $A(x)$. In one case it is necessary to use a tangent construction to verify that an interval does not contain any roots. In this section we present several variants of the Collins-Loos algorithm, which are derived using Theorem 28 and Fourier's theorem.

The key to obtaining an algorithm using Fourier's theorem is to guarantee that the intervals that are tested do not contain any roots of the derivative. In this case, Fourier's theorem reduces to a combination of the intermediate value theorem and Rolle's theorem: if $\forall x \in [a, b], A^{(1)}(x) \neq 0$, then the number of roots of $A(x)$ in $(a, b]$ is equal to $\text{var}(\text{sign}(A(a)), \text{sign}(A^{(1)}(a))) - \text{var}(\text{sign}(A(b)), \text{sign}(A^{(1)}(b)))$. In particular, if $\alpha'_1 < \dots, < \alpha'_r$ are the distinct roots of $A^{(1)}(x)$ then the intermediate value theorem

can be applied to the intervals $(\alpha'_i, \alpha'_{i+1}]$ to find the isolating intervals for the simple roots $A(x)$. It is necessary to consider the possibility of multiple roots since even if $A(x)$ is squarefree, $A^{(1)}(x)$ may not be (e.g. $x^n + 1$). For the multiple roots it is necessary to determine if $A(\alpha'_i) = 0$. The algorithm **ROLLE**, listed in Figure 31, recursively uses the derivative sequence and these ideas to find isolating intervals for $A(x)$.

$$L \leftarrow \mathbf{ROLLE}(A(x))$$

[Isolate real roots using Rolle's theorem and the derivative sequence. $A(x)$ is an integral polynomial. $L = (I_1, m_1, \dots, I_r, m_r)$ a list of isolating intervals for the real roots of $A(x)$. m_j is the multiplicity of the root in I_j .]

1. [Initialize.] $B \leftarrow \mathbf{RootBound}(A(x)); L \leftarrow ()$.
2. [Recursion.]
 - if $\deg(A(x)) = 1$ then {
 - $L := ((-B, B), 1)$; return }
 - else { $L' \leftarrow \mathbf{ROLLE}(A'(x)); r' \leftarrow \text{Length}(L')/2$ }.
3. [Compute $\text{sign}(A(\alpha'_i))$ and apply intermediate value theorem.]
 - Let $\alpha'_1 < \dots < \alpha'_{r'}$ be the distinct roots of $A'(x)$;
 - Let $m'_1 < \dots < m'_{r'}$ be the multiplicities of α'_i ;
 - $\alpha_0 \leftarrow -B$; $\alpha_{r'+1} \leftarrow B$;
 - for $i = 1, \dots, r' + 1$ do {
 - if $\text{sign}(A(\alpha'_i)) = 0$ then
 - $L \leftarrow \text{append}(L, I_i, m'_i + 1)$
 - else if $\text{sign}(A(\alpha'_{i-1})) \cdot \text{sign}(A(\alpha_i)) < 0$ then
 - $L \leftarrow \text{append}(L, (\alpha_{i-1}, \alpha_i), 1)$ } ■

Figure 31: Algorithm **ROLLE**

This algorithm requires the computation of the sign of a real algebraic number. In particular we must determine the sign of a polynomial at the real roots of its

derivative where the roots of the derivative are given by isolating intervals. Let α' be a root of $A^{(1)}(x)$ and let $I' = (a, b]$ be an isolating interval for α' . If $C(x) = \text{gsfd}(\text{gcd}(A(x), A^{(1)}(x)))$, then $A(\alpha') = 0$ if and only if $C(\alpha') = 0$. Since any root of $C(x)$ is a root of $A^{(1)}(x)$, the only possible root of $C(x)$ in the interval I' is α' . Moreover, since $C(x)$ is squarefree it only has simple roots and $C(\alpha') = 0$ if and only if $C(a)C(b) < 0$ or $C(b) = 0$.

If $A(\alpha') \neq 0$ then its sign can be computed by repeatedly bisecting I' , always retaining the subinterval that contains α' until I' does not contain any roots of $A(x)$. If I' does not contain any roots of $A(x)$ then $\text{sign}(A(\alpha'))$ is equal to the sign of any number in I' . In particular $\text{sign}(A(\alpha')) = \text{sign}(A(b))$. The bisection process must eventually terminate since $A(\alpha') \neq 0$. Any algorithm which can determine the number of real roots in an interval can be used to test if I' contains any roots of $A(x)$. However, all that is required is an algorithm which can determine if there are any roots of $A(x)$ in I' .

If I is an isolating interval for a root, α , of $A(x)$, **AFSIGN**($A(x), I, B(x); s, J$) returns $\text{sign}(B(\alpha))$ and a subinterval, J , of I which does not contain any roots of $B(x)$. **AFSIGN** uses Descartes' rule of signs and a series of polynomial transformations to find a subinterval of I that does not contain any roots of $B(x)$. The sign of $B(\alpha)$ is equal to the sign of any number contained in J . The computation is performed by selecting a rational number in J . For a more complete description of **AFSIGN** see Section 5.3.

The algorithm **ROLLE** produces isolating intervals $(\alpha'_i, \alpha'_{i+1}]$ which may not be

rational intervals. Rational intervals can be produced as a side effect of the sign computation just described. Suppose $(\alpha'_i, \alpha'_{i+1}]$ is an isolating interval for a root, α , of $A(x)$, and $J_i = (a_i, b_i]$ is an isolating interval for α'_i which does not contain any roots of $A(x)$. Then $(b_i, a_{i+1}]$ is an isolating interval for α .

A further practical improvement to the algorithm is obtained by using the primitive derivative sequence rather than the derivative sequence. This greatly reduces the size of the coefficients.

Theorem 50 $|\text{pp}(A^{(k)}(x))|_i \leq \binom{m}{k} |A(x)|_i$.

PROOF.

$$\begin{aligned} A^{(k)}(x) &= \sum_{i=0}^{m-k} \frac{(m-i)!}{(m-i-k)!} a_{m-i} x^{m-k-i} \\ &= k! \sum_{i=0}^{m-k} \binom{m-i}{k} a_{m-i} x^{m-k-i}. \end{aligned}$$

Therefore, $k!$ divides $A^{(k)}(x)$ and $|\text{pp}(A^{(k)}(x))|_i \leq \binom{m}{k} |A(x)|_i$ ■

4.3.1 Tangent Construction

Since the algorithm **ROLLE** inductively uses the derivative, specialized algorithms can be designed to compute the sign of $A(x)$ at the roots of its derivative. Rolle's theorem implies that between any two distinct roots of $A(x)$ there is a root of $A^{(1)}(x)$. This implies that there can be only zero, one, or two roots of $A(x)$ in an isolating interval for a simple root of $A^{(1)}(x)$. It further implies that between any two isolating intervals for $A(x)$ there is an isolating interval for $A^{(1)}(x)$ and hence the isolating intervals produced by the algorithm **ROLLE** are separated. The intervals $I_j = (a_j, b_j]$

and $I_k = (a_k, b_k]$ are separated if either $b_j < a_k$ or $b_k < a_j$. Since the isolating intervals for $A^{(1)}(x)$ are inductively separated from the isolating intervals for $A^{(2)}(x)$, $A^{(2)}(x)$ does not vanish on any isolating interval, I' , for a simple root of $A^{(1)}(x)$, which implies, in this case, that $A^{(1)}(x)$ is monotone on I' . If $A^{(1)}(x)$ is monotone on the interval $I = (a, b]$ then the tangent to $A(x)$ at a and the tangent to $A(x)$ at b do not intersect $A(x)$ for x in the interior of I and they can be used to determine if $A(x)$ has any roots in I .

Lemma 11 (Tangent Construction) *Assume that α' is a root of $A^{(1)}(x)$ of multiplicity m' , and that $I' = (a, b]$ is an isolating interval for α' . Further assume that $A^{(2)}(x) \neq 0$ for all $x \in I$ except possibly for α' . Let $T_a(x)$ be the tangent to $A(x)$ at a and let $T_b(x)$ be the tangent to $A(x)$ at b .*

1. *If $A(a)A^{(1)}(a) > 0$ and $A(b)A^{(1)}(b) < 0$ then $A(x)$ does not have any roots in I .*
2. *If m' is even then $A(x)$ has at most one real root in I .*
3. *Let $a' = a - A(a)/A^{(1)}(a)$ be the intersection of $T_a(x)$ and the x -axis. Similarly let $b' = b - A(b)/A^{(1)}(b)$ be the intersection of $T_b(x)$ and the x -axis. If $a' > b'$ then $A(x)$ does not have any roots in I .*

PROOF. Part 1 follows from Lemma 5. $A(x)$ and $A^{(1)}(x)$ are monotone on $(a, \alpha']$ and $(\alpha', b]$. If m' is even $A(x)$ is monotone on I and $A(x)$ can have at most one root in I , which proves Part 2. If m' is odd then $A^{(1)}(x)$ is monotone on I . Without loss of generality we can assume that $A^{(m'+1)}(x) > 0$. Let $0 < x < b - a$. By the mean value theorem $A(x + a) = A(a) + A^{(1)}(\gamma)x$ for some $a < \gamma < b$. Since $A^{(1)}(x)$ is monotone

increasing $A(x + a) > A(a) + A^{(1)}(a)x = T_a(x + a)$. Similarly $A(b - x) > T_b(b - x)$. Therefore, if $a' > b'$, then $A(x) > 0$ for $x \in I$ ■

The algorithm **IPRRSTC**, listed in Figure 32, uses this lemma to bisect an isolating interval, I' for a root of $A^{(1)}(x)$ until it contains no roots of $A(x)$. If m' is even then **IPRRSTC** uses the subalgorithm **IPRRS** to find a subinterval which does not contain any roots of $A(x)$. Otherwise Parts 1 and 2 are used to check if I' contains any roots of $A(x)$. If it can not be determined that $A(x)$ does not have any roots in the interval I' , I' is bisected and the test is performed again. The tangent construction will eventually prove that there are no roots of $A(x)$ in a subinterval of I' since if the length of I' is small enough, $T_a(\alpha')$ and $A(\alpha')$ have the same sign. This follows from the Taylor series expansion of $A(x)$ about a . For a detailed proof along with a bound on how small I' must be see [14].

The intersection, a' , of the tangent $T_a(x)$ and the x -axis can be computed in time dominated by $(mL(a) + L(|A(x)|_\infty))^2$. A general rational quotient is not required to compute $A(a)/A^{(1)}(a)$ since $A(e/f)/A^{(1)}(e/f) = f^m A(e/f)/f \cdot f^{m-1} A'(e/f)$, where $a = e/f$.

4.3.2 Improvement Using Fourier's Theorem

A bound on the number roots of $A(x)$ in an isolating interval for $A^{(1)}(x)$ can be obtained using Fourier's theorem. In some cases this is enough information to determine the number of roots of $A(x)$ in the interval and the general purpose sign algorithm or tangent construction can be avoided. The case where the tangent construction or general sign computation is required corresponds to the case where Collins and Loos

$$J \leftarrow \text{IPRRSTC}(A'(x), I', m', A(x))$$

[Integral polynomial relative root separation using tangent construction. The inputs are $A'(x)$, I' , m' , and $A(x)$. $A(x)$ is an integral polynomial and $A'(x)$ is its derivative. I' is a left-open right-closed binary rational isolating interval for a root, α' of $A'(x)$ of multiplicity m' . I' does not contain any roots of $A^{(2)}(x)$ except possibly α' . The output J is a subinterval of I' which does not contain any roots of $A(x)$.]

1. [Check if m' is even.]
if even(m') then { $J \leftarrow \text{IPRRS}(A'(x), I', A(x))$; return }.
2. [Tangent construction.] $a \leftarrow \text{LeftEndpoint}(I')$; $b \leftarrow \text{RightEndpoint}(I')$; $s_1 \leftarrow \text{sign}(A(a))$; $s'_1 \leftarrow \text{sign}(A'(a))$; $t_1 \leftarrow \text{sign}(A(b))$; $t'_1 \leftarrow \text{sign}(A'(b))$; if $s'_1 \neq 0$ then $a' \leftarrow \text{IUPBEI}(A(x), a) / (\text{num}(a) \text{IUPBEI}(A'(x), a))$ else $a' \leftarrow a$;
if $t'_1 \neq 0$ then $b' \leftarrow \text{IUPBEI}(A(x), b) / (\text{num}(b) \text{IUPBEI}(A'(x), b))$ else $b' \leftarrow b$;
while $a' < b' \wedge (s_1 s'_1 < 0 \vee t_1 t'_1 > 0)$ do {
 $c \leftarrow \text{RIB}((a, b))$; $w_1 \leftarrow \text{sign}(A(c))$; $s'_1 \leftarrow \text{sign}(A'(c))$;
 $c' \leftarrow \text{IUPBEI}(A(x), c) / (\text{num}(c) \text{IUPBEI}(A'(x), c))$;
 if $w'_1 t'_1 < 0 \vee t'_1 = 0$ then {
 $a \leftarrow c$; $a' \leftarrow c'$; $s_1 \leftarrow w_1$; $s'_1 \leftarrow w'_1$ }
 else {
 $b \leftarrow c$; $b' \leftarrow c'$; $t_1 \leftarrow w_1$; $t'_1 \leftarrow w'_1$ } };
 $J \leftarrow (a, b]$ ■

Figure 32: **IPRRSTC** Integral Polynomial Relative Root Separation Using Tangent Construction

required the tangent construction.

In the algorithm **ROLLE**, $A^{(m'+1)}(x) \neq 0$ for all x in I' , an isolating interval for a root α' of $A^{(1)}(x)$ of multiplicity m' . If $V_x = \text{var}(\text{sign}(A(x)), \dots, \text{sign}(A^{(m'+1)}(x)))$ and $V'_x = \text{var}(\text{sign}(A^{(1)}(x)), \dots, \text{sign}(A^{(m'+1)}(x)))$, then $V'_a - V'_b = m'$ and $V_a - V_b$ can be computed from $V'_a - V'_b$. Recall that there are zero, one, or two roots of $A(x)$ in I' . If $V_a - V_b = 0$, then there are no roots of $A(x)$ in I' . If $V_a - V_b$ is odd then there is one root of $A(x)$ in I' and **IPRRS** can be used to obtain a subinterval of I' not containing any roots of $A(x)$. Finally, if $V_a - V_b > 0$ and even, then $A(x)$ can have either zero or two roots in I' and either **IPRRSTC** or **AFSIGN** must be used to obtain a subinterval of I' containing no roots of $A(x)$.

$V_a - V_b$ can be computed from $V'_a - V'_b = m'$ using the inductive step used in the proof of Lemma 6.

$$V_a - V_b = \begin{cases} V'_a - V'_b + 1 & \text{if } A(a)A^{(1)}(a) < 0 \wedge A(b)A^{(m'+1)}(b) > 0. \\ V'_a - V'_b - 1 & \text{if } A(a)A^{(1)}(a) > 0 \wedge A(b)A^{(m'+1)}(b) < 0. \\ V'_a - V'_b & \text{otherwise} \end{cases}$$

The $\text{sign}(A^{(m'+1)}(b)) = \text{sign}(A^{(m'+1)}(a))$ which is equal to $-1^{m'}\text{sign}(A^{(1)}(a))$.

Algorithm **IPRIDSF** (Integral Polynomial Root Isolation using the Derivative Sequence and Fourier's theorem) is based on the algorithm **ROLLE** and uses this test based on Fourier's theorem and the general purpose algebraic sign algorithm **AFSIGN**. **AFSIGN** requires that the input interval be standard. Since this is not the case in general, the algorithm **IPSIFI** is called to first convert the interval to a standard interval. The algorithm **IPRIDTCF** (Integral Polynomial Root Isolation using the Derivative Sequence, Tangent Construction, and Fourier's theorem) is the same as **IPRIDSF** except that it uses the tangent construction algorithm **IPRRSTC**

for the case when $A(x)$ can have either zero or two roots.

4.3.3 An Example

In this section we compare the algorithm **IPRIDSF** and **IPRIDTCF** on the example from Section 4.1.3. Both algorithms use the primitive derivative sequence which can be compared to the primitive Sturm sequence used by **IPRIST**. The maximum number of bits of any coefficient in the derivative sequence is 27 compared to 123 in the Sturm sequence, and the average number of bits in the max norms of the polynomials in the derivative sequence is 20.7 compared to 60.25 in the Sturm sequence.

$$\begin{aligned} \text{pp}(A^0(x)) = & 96040x^7 + 105644x^6 - 4373838x^5 - 16035215x^4 - 33253965x^3 - \\ & 44130844x^2 - 30498902x - 7859280 \end{aligned}$$

$$\begin{aligned} \text{pp}(A^1(x)) = & 672280x^6 + 633864x^5 - 21869190x^4 - 64140860x^3 - 99761895x^2 - \\ & 88261688x - 30498902 \end{aligned}$$

$$\begin{aligned} \text{pp}(A^2(x)) = & 2016840x^5 + 1584660x^4 - 43738380x^3 - 96211290x^2 - \\ & 99761895x - 44130844 \end{aligned}$$

$$\text{pp}(A^3(x)) = 672280x^4 + 422576x^3 - 8747676x^2 - 12828172x - 6650793$$

$$\text{pp}(A^4(x)) = 96040x^3 + 45276x^2 - 624834x - 458149$$

$$\text{pp}(A^5(x)) = 6860x^2 + 2156x - 14877$$

$$\text{pp}(A^6(x)) = 70x + 11$$

There are four real roots of $A^{(1)}(x)$, $\alpha'_1 \in (-5, -4]$, $\alpha'_2 \in (-3/2, -1]$, $\alpha'_3 \in$

$(-7/8, -3/4]$, and $\alpha'_4 \in (6, 7]$. Since $\text{sign}(-\infty) = -1$, $\text{sign}(\alpha'_1) = 1$, $\text{sign}(\alpha'_2) = 1$, $\text{sign}(\alpha'_3) = 1$, $\text{sign}(\alpha'_4) = -1$, and $\text{sign}(\infty) = 1$, $A(x)$ has three real roots. **IPRIDSF** required three, one, three, and two bisections to determine these signs, while **IPRIDTCF** required three, two, three, and two bisections to determine the signs. The number of bisections for **IPRIDSF** includes the number of bisections for **IPSIFI** to convert the interval to a standard interval.

Recursively, the number of real roots of $A^{(2)}(x)$ is three, $A^{(3)}(x)$ is two, $A^{(4)}(x)$ is three, $A^{(5)}(x)$ is two, and $A^{(6)}(x)$ is one. For the 15 real roots of $A^{(1)}(x)$ through $A^{(6)}(x)$, $V_a - V_b = 0$ one time, $V_a - V_b = 1$ ten times, and $V_a - V_b = 2$ four times. The only difference between the two algorithms is the way the signs are determined for the four cases where $V_a - V_b = 2$. In every case except one, both algorithms (**IPRIDSF** and **IPRIDTCF**) required the same number of signs: two, one, and six. In the case where the number of bisections was different, **IPRIDSF** required one bisection and **IPRIDTCF** required two. The average number of bisections per sign computation was 1.66 for **IPRIDSF** and 1.72 for **IPRIDTCF**. The maximum number of bisections was six in both cases.

4.4 Computing Time Analysis

In this section we derive maximum computing time bounds for each of the real root isolation algorithms introduced in this chapter. The algorithm for which we were able to derive the best maximum computing bound is the algorithm based on polynomial transformations and Descartes' rule of signs, **IPRICS**. The next best bound is for the derivative sequence algorithm, and the worst bound is for the Sturm sequence

algorithm. Even though these theorems are not indicative of typical computing times, they do suggest which algorithms are better in practice. We conclude by giving an example which shows that the computing time bound for the algorithm based on Descartes' rule of signs is essentially the best possible. Unfortunately, this example does not exhibit the maximum computing times for the other two algorithms. Nonetheless, the theoretical computing times for those algorithms on this example are larger than theoretical computing time for **IPRIS**.

Theorem 51 (Computing Time of **IPRIS)** *Let $m = \deg(A(x))$ and $|A(x)|_1 = d$. Then the computing time of **IPRIS**($A(x)$) is dominated by $m^6 L(d)^3$.*

PROOF. The proof involves estimating the computing time associated with each node in the search tree and obtaining a bound on the height and number of nodes in the tree.

At level h the endpoints of each interval are of the form $jb/2^i$ with $i \leq h$, $0 \leq j < 2^h$, and b a root bound. Therefore the length of any endpoint is less than or equal to $L(b) + h$, which by Theorem 8 is dominated by $L(d) + h$. By Theorem 2, the height of the tree (the maximum value of h) is dominated by $mL(d)$.

An interval at level h is bisected if and only if it contains two or more real roots. Let k be the number of intervals that are bisected at level h . Since the length of any interval at level h is equal to $b/2^h$ an interval is bisected if and only if it contains two real roots whose separation is less than $b/2^h$. Therefore, the product of the separations of the real roots in the intervals to be bisected is less than $b^k/2^{hk}$ (we can assume $b/2^h < 1$). However, by Theorem 3 this product is greater than $c_0 d^{-c_1 m}$, for

some constants c_0 and c_1 , which implies that $hk \preceq mL(d)$.

The cost of **IPRIST** at level h is dominated by the cost of the k evaluation of the Sturm sequence. By Theorem 48 the cost of each evaluation at a point a is dominated by $m^3L(d)L(a) + m^3L(a)^2$, which since $L(a) \preceq L(d) + h$ is dominated by $m^3L(d)^2 + m^3L(d)h + m^3L(d)^2 + m^3h^2$. The k evaluations at level h are dominated by $m^3kL(d)^2 + hk(m^3L(d) + m^3h)$, which is dominated by $m^4L(d)^2 + m^4L(d)^2 + m^5L(d)^2 \preceq m^5L(d)^2$. Since there are at most $mL(d)$ levels the total computing time of **IPRIST** is dominated by $m^6L(d)^3$ ■

A better computing time bound can be obtained for the algorithm which uses the recurrence relation method for evaluating Sturm sequences.

Theorem 52 (Computing time of IPRIST using IPRSEVS.)

*Let $m = \deg(A(x))$ and $|A(x)|_1 = d$. Then the computing time of **IPRIST**($A(x)$) is dominated by $m^5L(d)^3$.*

PROOF. The proof is the same as Theorem 51 except Theorem 49 is used for the time to evaluate the Sturm sequence ■

The same technique used in the proof of Theorem 51 can be used to derive a computing time bound for **IPRICS**. However, unlike the Sturm sequence algorithm, complex roots have an effect on whether an interval is bisected or not, and the corollary of Davenport's theorem which deals with pairs of complex conjugate roots must be used.

Theorem 53 (Computing time of IPRICS) *Let $m = \deg(A(x))$ and $|A(x)|_1 = d$. Then the computing time of **IPRICS**($A(x)$) is dominated by $m^5L(d)^2$.*

PROOF. The cost associated with a node in the search tree is dominated by the cost of the translation. By Theorem 20 the max norm of a polynomial associated with a node at level h is dominated by $2^{mh}d$. Therefore, Theorem 35 implies that the time for the translation is dominated by $m^3h + m^2L(d)$.

An interval associated with a node at level h has length equal to $b/2^h$, where b is the root bound used by the algorithm. By Theorem 27 an interval at level h is bisected only if it contains two or more real roots or there is a pair of complex conjugate roots in the circle of radius 2^{h+1} centered about the interval in question. Therefore an interval is bisected only if it contains two real roots whose separation is less than $b/2^h$ or if the surrounding circle contains a pair of complex conjugate roots whose separation is less than $b/2^{h-1}$. Let k be the number of intervals that are bisected at level h . Then the product of the separations of the roots to be bisected and the interfering complex conjugate roots is less than $b^k/2^{hk}$. However, by Corollary 3 and 4, this product is greater than $c_0(d)^{-c_1m}$, for some constants c_0 and c_1 , which implies that $hk \preceq mL(d)$.

The cost of the k translations at level h is dominated by $m^3hk + km^2L(d)$, which is dominated by $m^4L(d)$. By Theorem 2 the height of the tree is dominated by $mL(d)$ and hence there are at most $mL(d)$ levels. Therefore, the total computing time of **IPRICS** is dominated by $m^5L(d)^2$ ■

The proof of the maximum computing time bound for the derivative sequence algorithms depends on the time required to compute the sign of a real algebraic number. In particular, we must bound the cost for computing the signs of all of the

elements of the derivative sequence at the roots of their derivatives.

Theorem 54 (Computing Time of IPRIDS) *Let $A(x)$ be an integral polynomial with $\deg(A(x)) = m$ and $|A(x)|_1 = d$. Then the computing time of $\text{IPRIDS}(A(x))$ is dominated by $m^8 + m^6 \log(d)^2$.*

PROOF. The computing time is dominated by the time to compute the signs of $A^{(k)}(x)$ at the roots of $A^{(k+1)}(x)$. By Theorem 50, $L(|A^{(k)}(x)|_1) \preceq m + L(d)$. Therefore, by Theorem 75, the signs of $A^{(k)}(x)$ at all of the real roots of $A^{(k+1)}(x)$ can be computed in time dominated by $m^5(m + L(d))^2$, which is dominated by $m^7 + m^5 L(d)^2$. Since there are m derivatives of positive degree the theorem is proved ■

If instead the tangent construction is used to compute the signs of the polynomials at the roots of their derivatives, the maximum computing time increases.

Theorem 55 (Computing Time of IPRIDTC) *Let $A(x)$ be an integral polynomial with $\deg(A(x)) = m$ and $|A(x)|_1 = d$. Then the computing time of IPRIDTC is dominated by $m^9 + m^6 \log(d)^3$.*

PROOF. The proof is the same as Theorem 54 except the cost of computing the signs with the tangent construction is $m^8 + m^5 L(d)^3$, since the cost of each tangent construction is dominated by $(mL(a) + L(|A^{(k)}(x)|_1))^2$, $L(a) \preceq mL(|A^{(k)}(x)|_1)$, and there are at most $mL(|A^{(k)}(x)|_1)$ tangent constructions per derivative ■

The maximum computing times are typically much larger than those obtained in practice since the roots of a typical polynomial are much further apart than the bound obtained in Theorem 2. However, it is possible to show that these computing time

bounds can not be improved since there are polynomials of degree m with coefficients of size $L(d)$ that require computing time codominant with the times in these theorems.

The following example, due to Mignotte [38] can be used to obtain the maximum computing time for **IPRICS** in Theorem 53. The polynomial $A(x) = x^m - 2(ax - 1)^2$ has two real roots in an interval of length $2a^{-(m+2)/2}$ and hence the height of the search tree required by either **IPRIST** or **IPRICS** is dominated by $mL(a)$. Also since $A^{(1)}(x)$ has a real root between these two roots, the number of bisections required to compute the signs in the algorithm **IPRIDS** or **IPRIDTC** is also dominated by $mL(a)$.

Theorem 56 (Mignotte) *Let $A(x) = x^m - 2(ax - 1)^2$, $m \geq 3$ and $a \geq 2$ be an integer, then $A(x)$ has two real roots in the interval $(1/a - h, 1/a + h)$ where $h = a^{-(m+2)/2}$.*

PROOF. $A(1/a \pm h) = (1/a \pm h)^m - a^2 h^2$, which is less than $(1/a)^m \sum_{i=0}^m (mh/a)^i$. Since $m \geq 3$ and $a \geq 2$, $(mh/a) < 1/2$ and the geometric sum is equal to 2. Therefore $A(1/a \pm h) < 2(1/a)^m - 2(a/a)^m = 0$. Since $A(1/a) = (1/a)^m > 0$, $A(x)$ has at least two real roots in the interval $(1/a - h, 1/a + h)$ ■

Since $\text{var}(A(x)) = 3$, Descartes' rule of signs implies that $A(x)$ has three positive roots, and since $\text{var}(A(-x)) = 1$ if m is even and $\text{var}(A(-x)) = 0$ if m is odd, $A(x)$ has one or zero negative roots depending on whether m is even or odd. Moreover, using the prime 2, Eisenstein's criteria shows that $A(x)$ is irreducible.

Theorem 57 (Maximum Computing Time) *Let $A(x) = x^m - 2(ax - 1)^2$ with $m \geq 3$ and $a \geq 2$ an integer. Then either the computing time of **IPRICS**($A(x)$) or*

the computing time of $\text{IPRCS}(A(-x+1))$ is codominant with $m^5L(a)^2$.

PROOF. By Theorem 56 the height of the tree associated with IPRCS of $A(x)$ is codominant with $mL(a)$. The polynomial associated with node (l, n) is $2^{ml}A(x + n/2^l) = (x+n)^m - 2^{ml-1}a(x+n)^2 + 2^{ml+1}(x+n) - 2^{ml}$. This polynomial is dense and has max norm codominant with $L(l)$. However, if n is small most of the coefficients will have small sizes. To guarantee that all of the coefficients have large sizes, we want to ensure that the search tree has nodes with large n . If all of the nodes of corresponding to $A(x)$ are such that $n < 2^{l-1}$ the transformed polynomial $A(-x+1)$ will have nodes with $n > 2^{l-1}$. Therefore, for either $A(x)$ or $A(-x+1)$ the search tree will have nodes with polynomials that are dense and have coefficients codominant with $L(l)$ which is codominant with $mL(a)$. The computing time for the translation associated with such a node is $m^3l + m^2L(a)^2$ which is codominant with $m^4L(a)$. Since there are at least $mL(a)$ nodes, the total computing time is codominant with $m^5L(a)^2$ ■

If $A(-x-1)$ is required this does not provide a worst case example since $L(|A(-x-1)|_1) \sim m + L(a)$. This is what is meant when we say the computing time is essentially the best that can be obtained.

The computing time for the Sturm sequence algorithm on this class of polynomials is $m^5L(a)^3$ since the cost of evaluating the Sturm sequence at a point of size $mL(a)$ is $m^4L(a)^2$ and there are at least $mL(a)$ such evaluations. This is not the maximum computing time obtained in Theorem 51. This example does not produce a worst case for the Sturm sequence algorithm since there are only five polynomials in the Sturm

sequence. Similarly this does not produce a worst case example for the derivative sequence algorithm since $\text{pp}(A^{(k)}(x)) = x^{m-k}$ for $k > 2$. Nonetheless, the computing time the time required to compute the sign of the root of $A^{(1)}(x)$ between the two roots of $A(x)$ in the interval $(1/a - h, a/a + h)$ requires time codominant with $m^5L(a)^2$ if Descartes' rule is used to compute the sign and $m^5L(a)^3$ if the tangent construction method is used.

4.5 Empirical Behavior and Average Computing Time

This section contains an empirical comparison of the Sturm sequence algorithm, the derivative sequence algorithm, and the coefficient sign variation method. All three algorithms were executed on a large number of examples from several classes of polynomials. These classes include Mignotte's example, polynomials with random coefficients, polynomials with varying number of random real roots, and orthogonal polynomials. Further examples such as random bivariate resultants and discriminants are discussed in chapter VII. For almost all of these examples the coefficient sign variation method is far superior to the other two algorithms, even for orthogonal polynomials which have Sturm sequences with unusually small coefficients. Nonetheless, there are examples for which the Sturm sequence algorithm is superior.

4.5.1 Exceptional Polynomials

In this section we examine the performance of the different root isolation algorithms on Mignotte's class of polynomials. We also discuss a related example for which the Sturm sequence algorithm is superior to the other two algorithms.

Tables 7 and 8 report the computing times and search tree information for the Sturm sequence algorithm **IPRIST** and the coefficient sign variation method **IPRICS** for the degree ten Mignotte polynomials $A(x) = x^{10} - (3^k x - 1)^2$, with k ranging from 5 to 9. H is equal to the maximum of the heights of the search trees for the positive and negative roots, N is equal to the sum of the number of nodes in the search trees for the positive and negative roots, and L is equal to the sum of the number of leaf nodes in the search trees for the positive and negative roots. The computing times are in milliseconds and are listed in the column under the algorithm's name. For this example, the tables show that the algorithms behave similarly in terms of the number of bisections they must perform. Initially the computing time of the Sturm sequence algorithm is faster than the coefficient sign variation method; however, as k increases, the coefficient sign variation eventually becomes faster. The reason for this, is that the polynomial evaluations that the Sturm sequence algorithm performs become slower compared to the translation required by the coefficient sign variation method as the height of the tree increases.

Table 7: **IPRIST** Statistics for $x^m - (ax - 1)^2$

m	k	r	H	N	L	IPPNPRS	IPRIST
10	5	4	51	102	52	0	167
10	6	4	102	204	103	33	650
10	7	4	150	300	151	33	1567
10	8	4	201	402	202	33	1567
10	9	4	246	492	247	33	5183

Let $A(x) = x^m + (ax - 1)^2$. These polynomials do not have any positive real roots

Table 8: **IPRICS** Statistics for $x^m - (ax - 1)^2$

m	a	r	H	N	L	IPRICS
10	5	4	51	102	52	334
10	6	4	102	204	103	1000
10	7	4	150	300	151	1917
10	8	4	201	402	202	3366
10	9	4	246	492	247	5017

since $A(x) > 0$ for $x > 0$. The polynomial $A(x)$ has a pair of complex conjugate roots close to $1/a$ since it is a perturbation of $(ax - 1)^2$ which has a multiple at $1/a$. The multiple root must transform into a pair of complex conjugate roots since $A(x)$ does not have any positive real roots. The derivative sequence algorithm and the coefficient sign variation method behave poorly on this example, while the Sturm sequence algorithm behaves well. The reason for the poor performance of the coefficient sign variation method, is that it continues transforming the polynomial until the pair of complex conjugate roots have been sufficiently separated. Similarly, the derivative sequence algorithm must perform many bisections to determine the sign of $A(x)$ at the root of its derivative which is close to $1/a$, since $A(x)$ is nearly tangent to the x -axis at this root. The Sturm sequence algorithm, however, is not affected by complex roots and hence determines that there are no positive roots immediately. The Sturm sequence algorithm also benefits from that fact that $A(x)$ only has five polynomials in its Sturm sequence.

Tables 9 and 10 report the computing times and search tree information for

the Sturm sequence algorithm **IPRIST** and the coefficient sign variation method **IPRICS** for the degree ten polynomials $A(x) = x^{10} + (3^k x - 1)^2$, with k ranging from 5 to 9. For this example, the Sturm sequence algorithm is always significantly faster than the coefficient sign variation method. The height and number of nodes for the coefficient sign variation method is similar to the previous example; however, the number of nodes is always equal to 2 for the Sturm sequence algorithm. The reason for this, as pointed out previously is that the Sturm sequence is not affected by the pair of complex conjugate roots near $1/a$ as is the coefficient sign variation method.

Table 9: **IPRIST** Statistics for $x^m + (ax - 1)^2$

m	a	r	H	N	L	IPRISS	IPISRS
10	3^5	0	1	2	2	0	0
10	3^6	0	1	2	2	0	17
10	3^7	0	1	2	2	0	33
10	3^8	0	1	2	2	0	33
10	3^9	0	1	2	2	0	50

Table 10: **IPRICS** Statistics for $x^m + (ax - 1)^2$

m	a	r	H	N	L	IPRICS
10	3^5	0	51	100	51	350
10	3^6	0	97	194	98	1084
10	3^7	0	146	292	147	2133
10	3^8	0	194	388	195	3550
10	3^9	0	240	480	241	5332

4.5.2 Random Polynomials and Average Computing Time

In this section we investigate the average performance of the three real root isolation algorithms described in this chapter. Each algorithm was executed for 1500 randomly generated input polynomials with degrees from 10 to 100 and coefficient sizes from 10 to 250 bits. The average execution times are calculated and compared. This comparison clearly shows that, for random polynomials, the coefficient sign variation method is far superior to the other two methods, with the Sturm sequence method significantly worse than the various derivative sequence methods.

In our experiments, one thousand squarefree polynomials with uniformly distributed random integral coefficients were generated, and were used as input for each of the three algorithms. For each degree, m , and coefficient size, k , 100 random integral polynomials, whose coefficients are uniformly distributed between $-2^k + 1$ and $2^k - 1$, were generated. The computing times for each algorithm are averaged over the 100 random polynomials for each degree and coefficient size specifications. All times are reported in milliseconds and are listed under the corresponding algorithm's name. For each of the three algorithms several variations are compared.

Besides recording the computing times, a variety of statistics were gathered that characterize the computing times of the various algorithms. These statistics are also averaged over the 100 polynomials for each degree and coefficient size. For the Sturm sequence algorithm, **IPRIST**, the length of the Sturm sequence, l , the average number of bits in the Sturm sequence, S , and the maximum number of bits in the Sturm sequence, S_∞ , were computed. This information was also computed for

the Recurrence relation version of the Sturm sequence algorithm, **IPRISTR**. The average number of bits, T , were computed for the coefficient sign variation method, **IPRICS**, and its dual algorithm **IPRICSD**. Information concerning the search trees, such as height, H , number of nodes, N , and number of leaf nodes, L , were recorded for both **IPRIST** and **IPRICS**. H is the maximum of the heights of the search trees for the negative and positive roots. N and L are the sum of the nodes and leaf nodes of the search trees for the negative and positive roots.

The information characterizing the derivative sequence algorithms include the average number of bits in derivative sequence, the average number of real roots of the polynomials in the derivative sequence, and the average number of bisections per sign computation. The number of bits in the derivative sequence is compared to the number of bits in the primitive derivative sequence. For the improved algorithm **IPRIDSF**, which uses Fourier's theorem, the number of times $V_a - V_b$ is equal to zero, one, or two is recorded.

Table 11 reports on Sturm Sequence Calculation for 1000 randomly generated polynomials with degrees from 10 to 50 and coefficient sizes from 10 to 250 bits. For each degree and coefficient size equal to k , 100 random integral polynomials, whose coefficients are uniformly distributed between $-2^k + 1$ and $2^k - 1$, were generated. The average length of the Sturm sequence, the average number of bits in the primitive Sturm sequence (S) and quotient sequence (Q), the average maximum number of bits in the primitive Sturm sequence (S_m) and quotient sequence (Q_m), and times computing times for **IPPNPRS** and **IPPSQSEQ** are recorded.

Table 11: Sturm Sequence Statistics for Random Polynomials

m	k	l	S	S_m	IPNPRS	Q	Q_m	IPPSQSEQ
10	10	11	73.1	174.5	109.8	143.4	327.0	119.0
20	10	21	191.9	427.8	1460.2	380.8	833.1	1569.1
30	10	31	320.0	693.3	7064.4	636.7	1364.7	7547.3
40	10	41	455.3	975.4	22209.8	907.3	1927.7	22704.08
50	10	51	592.9	1257.3	54404.1	1182.3	2492.6	55256.5
10	50	11	342.4	815.6	1216.4	678.34	1529.17	1277.3
10	100	11	679.1	1615.4	4272.2	1347.18	3028.87	4429.19
10	150	11	1015.4	2415.7	9223.3	2015.30	4528.95	9713.84
10	200	11	1351.8	3215.3	16039.3	2683.65	6028.74	16923.05
10	250	11	1687.5	4013.5	24755.5	3350.35	7525.62	26018.95

Table 12 reports the time for isolating the real roots after a Sturm sequence is computed. The times for **IPIISS** which uses **IPLEV** for evaluation of Sturm sequences and **IPIISRS** which uses **IPRSEVS** for evaluation of Sturm sequences are reported. In addition the average number of real roots, r , the average height of the search trees, H , the average number of nodes in the search trees, N , and the average number of leaf nodes, L , in the search tree are reported. This data was obtained from the same random polynomials that were used for the experiments reported in Table 11.

Table 13 reports computing times and search tree statistics for **IPRICS** and **IPRICSD**. The table contains the same information concerning the search tree as was recorded for **IPRIST**. In addition, the average number of bits of the transformed polynomials was computed. T_1 is the average number of bits for **IPRICS** and T_2 is the average number of bits for **IPRICSD**.

Table 12: **IPHISS** Search Tree Statistics for Random Polynomials

m	k	r	H	N	L	IPHISS	IPHSRS
10	10	2.14	2.33	5.06	3.53	13.85	65.89
20	10	2.58	3.01	6.72	4.36	109.26	679.44
30	10	2.80	3.18	7.14	4.57	368.78	2569.21
40	10	2.92	3.32	7.76	4.88	933.14	14181.9
50	10	3.14	3.65	8.72	5.36	2041.13	14838.31
10	50	2.04	2.26	4.88	3.44	36.32	742.46
10	100	2.10	2.24	4.90	3.45	65.52	2650.99
10	150	1.92	2.11	4.56	3.28	87.66	5534.48
10	200	1.94	2.10	4.46	3.23	110.14	9537.05
10	250	1.96	1.94	4.10	3.05	127.47	14051.05

Table 13: **IPRICS** Search Tree Statistics for Random Polynomials

m	k	r	H	N	L	IPRICS	T_1	IPRICSD	T_2
10	10	2.14	3.41	8.94	5.47	25.78	20.44	29.23	24.51
20	10	2.58	3.94	10.92	6.46	40.55	56.51	48.80	76.39
30	10	2.80	3.98	11.52	6.76	54.41	111.17	67.47	141.51
40	10	2.92	4.31	12.60	7.30	73.38	192.99	91.87	234.68
50	10	3.14	4.57	13.36	7.68	89.95	306.96	114.98	362.17
60	10	3.10	4.29	12.60	7.30	97.43	404.50	127.35	466.33
70	10	3.05	4.09	12.50	7.25	112.58	554.46	146.17	620.98
80	10	3.28	4.89	14.48	8.24	142.50	919.37	183.98	971.53
90	10	3.38	4.85	14.54	8.27	153.65	1201.48	202.03	1235.04
100	10	3.74	5.10	15.20	8.60	175.39	1659.05	230.05	1658.39
10	50	2.04	3.27	8.46	5.23	65.80	27.35	69.15	31.02
10	100	2.10	3.31	8.60	5.30	113.01	33.06	116.57	36.17
10	150	1.92	3.17	8.06	5.03	163.40	36.55	166.91	40.51
10	200	1.94	3.03	7.54	4.77	213.53	39.48	216.79	42.03
10	250	1.96	2.87	7.34	4.67	263.37	44.16	266.65	45.83

Table 14 compares the computing times of several variations of the derivative sequence algorithm. Algorithms **IPRIDS**, **IPRIPDS**, and **IPRIDSF** use Descartes' rule of signs, algorithm **AFSIGN**, to compute the sign of the polynomial at the roots of its derivative. **IPRIDS** uses the derivative sequence, and **IPRIPDS** and **IPRIDSF** use the primitive derivative sequence. D is the average number of bits in the derivative sequence and D' is the average number of bits in the primitive derivative sequence. The algorithm **IPRIDSF** uses Fourier's theorem to test if $A(x)$ has zero, one, or two roots in the isolating intervals for the roots of its derivative. The algorithm **IPRIDTCF** also performs this test; however, if it needs to refine an isolating interval, this is done with the tangent construction. See Section 4.3 for more details concerning these variations.

Table 14: Comparison of Derivative Sequence Algorithms

n	k	D	IPRIDS	D'	IPRIPDS	IPRIDSF	IPRIDTCF
10	10	22.07	152.56	14.44	151.38	112.35	101.17
20	10	44.71	1118.18	21.47	921.82	644.03	589.33
30	10	71.12	4087.46	28.32	3345.96	2157.35	1945.32
40	10	100.23	10457.64	35.44	8428.08	4916.02	4396.89
50	10	131.08	23083.10	42.53	18445.36	9611.62	8479.45
60	10	163.64	43510.33	49.77	34231.45	16306.39	14237.53
70	10	197.18	78325.01	56.90	60643.33	26606.55	22774.57
10	50	62.10	258.67	54.50	266.21	227.41	235.33
10	100	112.13	337.77	104.47	366.05	324.19	362.47
10	150	162.18	400.22	154.57	448.17	412.82	486.83
10	200	212.08	506.37	204.53	570.20	524.71	653.28
10	250	262.06	583.84	254.43	671.64	623.31	792.25

Tables 15 and 16 report statistics characterizing the performance of the algorithms

IPRIDSF and **IPRIDTCF**. The average number of real roots, r , the average number of real roots of the polynomials in the derivative sequences, r' , and the average maximum number of real roots in the derivative sequence, r_0 are reported. This information indicates the number of sign computations and the maximum number of sign computations for any element of the derivative sequence. The number of bisections needed for the sign computations was also recorded. B is the average number of bisections per sign computation needed for each input polynomial. B_l is the average number of bisections per element of the derivative sequence. This is the number of bisections required to compute the signs of a polynomial at all of the roots of its derivative. B_∞ is the maximum of the number of bisections needed to compute the signs of a polynomial at any root of its derivative. V_0 , V_1 , and V_2 is the number of times Fourier's theorem indicated that there was zero, one, or two roots of $A(x)$ in the isolating intervals for the roots of its derivative.

The information in these tables can be used to conjecture average computing times for the different algorithms. This information is used along with a theorem due to Kac concerning the average number of real roots of a random polynomial, and an observation due to Knuth ([29]) that almost all pairs of polynomials have a normal PRS.

Theorem 58 (Kac) *The average number of real roots of a polynomial with uniformly distributed coefficients is asymptotic to $\log(m)$, where m is the degree of the polynomial.*

PROOF. [4] ■

Table 15: **IPRIDSF** Statistics for Random Polynomials

m	k	r	r'	r_0	B	B_l	B_∞	V_0	V_1	V_2
10	10	2.14	2.17	3.42	1.79	2.45	4.39	0.81	10.69	5.86
20	10	2.58	2.68	4.59	2.16	3.03	5.37	2.48	33.31	12.63
30	10	2.80	3.22	5.76	2.37	3.49	6.02	4.98	65.52	20.14
40	10	2.92	3.59	6.46	2.55	3.78	6.67	6.9	102.56	27.52
50	10	3.14	3.96	7.14	2.60	3.97	6.97	9.29	147.06	34.67
60	10	3.10	4.17	7.55	2.62	4.02	6.84	11.56	190.06	41.54
70	10	3.05	4.55	8.19	2.67	4.18	7.23	14.74	247.02	49.25
10	50	2.04	2.16	3.48	1.77	2.40	4.24	0.79	10.53	6.10
10	100	2.10	2.18	3.45	1.64	2.25	4.03	1.08	10.36	6.04
10	150	1.92	1.97	3.21	1.65	2.15	4.07	0.79	9.08	5.95
10	200	1.94	2.06	3.30	1.70	2.22	4.05	0.76	9.91	5.91
10	250	1.96	1.99	3.26	1.68	2.16	3.94	0.79	9.38	5.81

Table 16: **IPRIDTCF** Statistics for Random Polynomials

n	k	r	r'	R_0	B	B_l	B_∞	V_0	V_1	V_2
10	10	2.14	2.17	3.42	1.76	2.45	4.48	0.80	10.69	5.85
20	10	2.58	2.68	4.59	2.16	3.10	5.54	2.48	33.32	12.62
30	10	2.80	3.22	5.76	2.35	3.54	6.15	4.98	65.53	20.13
40	10	2.92	3.59	6.46	2.54	3.84	6.85	6.9	102.53	27.51
50	10	3.14	3.96	7.14	2.60	4.02	7.09	9.30	147.04	34.66
60	10	3.10	4.17	7.55	2.62	4.07	6.99	11.58	190.03	41.53
70	10	3.05	4.55	8.19	2.67	4.22	7.33	14.76	246.95	49.24
10	50	2.04	2.16	3.48	1.74	2.44	4.33	0.79	10.53	6.10
10	100	2.10	2.18	3.45	1.63	2.28	4.20	1.08	10.36	6.04
10	150	1.92	1.97	3.21	1.62	2.17	4.17	0.79	9.08	5.95
10	200	1.94	2.06	3.30	1.68	2.26	4.16	0.76	9.91	5.91
10	250	1.96	1.99	3.26	1.66	2.21	4.09	0.79	9.38	5.81

Knuth's observation about normal PRSs suggests that the Sturm sequence for a random polynomial should almost always be normal. The data in Table 11 agrees with this conjecture. Therefore, on average we can assume the computing time of **IPNPRS** is dominated by $m^4L(d)^2$. Since in practice the subresultant bound appears to be fairly accurate, this bound seems reasonable.

Table 12 suggest that the height of the search trees for the two calls of **IPISS** is proportional to the number of real roots. Theorem 58 implies that for random polynomials the number of real roots is dominated by $\log(m)$. The data in Table 12 agrees with this theorem. If the height of the search trees are dominated by $\log(m)$ then the size of the evaluation points are dominated by $L(RB(A(x)) + \log(m)$. By Theorem 8 $RB(A(x)) \preceq L(d)$; however, for random polynomials the root bound returned by **IUPRB** is usually equal to 2. This suggests that $L(a) \sim \log(m)$ and, assuming the subresultant bound for the Sturm sequence, the time for any Sturm evaluation with **IPLEV** is dominated by $m^3 \log(m)L(d) + m^3 \log(m)^2$.

The computing time of **IPISS** is dominated by the cost for the Sturm evaluations. The number of sturm evaluations is equal to the number of interior nodes which is equal to the number of nodes minus the number of leaf nodes. Table 12 suggests that this is proportional to the number of real roots, which by Kac's theorem is proportional to $\log(m)$. Therefore, under these assumptions the computing time of **IPRIST** is dominated by $m^4L(d)^2 + m^3 \log(m)^2L(d) + m^3 \log(m)^3 \preceq m^4L(d)^2$. This argument suggests the following conjecture for the average computing time of **IPRIST**.

Conjecture 1 (Average Computing Time of IPRIST) *Let $m = \deg(A(x))$ and $|A(x)|_\infty = d$. Assuming the coefficients of $A(x)$ are uniformly distributed between $-d$ and d , the average computing time of $\text{IPRIST}(A(x))$ is dominated by $m^4 L(d)^2$.*

This analysis agrees with the empirical results where the cost of computing the Sturm sequence is equal to $m^4 L(d)^2$ and clearly dominates the cost of searching for isolating intervals. In practise we can assume that $L(a) \sim 1$ so the time for evaluating the Sturm sequence at all midpoints should be proportional to $m^3 \log(m) L(d)$.

A similar argument can be carried out for the algorithm **IPRICS**.

Conjecture 2 (Average Computing Time of IPRICS) *Assume $A(x)$ is an integral polynomial with $\deg(A(x)) = m$ and $|A(x)|_\infty = d$. Assuming the coefficients of $A(x)$ are uniformly distributed between $-d$ and d , the average computing time of $\text{IPRICS}(A(x))$ is dominated by $m^3 \log(m)^2 + m^2 \log(m) \log(d)$.*

4.5.3 Polynomials with Varying Number of Real Roots

In the previous section we examined the behavior of the real root isolation algorithms for random polynomials. However, random polynomials may not be indicative of the polynomials one is interested in practice. Random polynomials typically have a small number of real roots. In practice this may not be the case. In this section we compare the algorithms on polynomials with a specified number of randomly generated real roots. In this experiment 10 random polynomials of degree 20 with r real roots were generated. A random polynomial with r real roots and s pairs of complex conjugate

roots is equal to

$$\left\{ \prod_{i=1}^r (a_i x + b_i) \right\} \left\{ \prod_{j=1}^s (c_j^2 x^2 - 2d_j c_j x + d_j^2 + e_j^2) \right\},$$

where a_i , b_i , c_j , d_j , and e_j are random integers uniformly distributed between $-2^k + 1$ and $2^k - 1$. In all of our experiments, $k = 5$. Since we compute the primitive part of the polynomial before isolating its real roots, we list the average number of bits, K , in the input polynomials. As the number of real roots increases, the average number of bits in the primitive parts decrease since there is a greater likelihood that a linear factor will have a non-trivial content. Each table in this section records the same information as was recorded in the section on random polynomials (Section 4.5.2).

Table 17: Sturm Sequence Statistics for Polynomials with Random Roots

n	K	r	l	S	S_m	IPNPRS	Q	Q_m	IPPSQSEQ
20	88.4	0	21	402.65	838.30	4789.9	725.17	1641.10	4921.5
20	86.8	2	21	399.46	828.40	4690.1	720.38	1618.70	4845.0
20	84.6	4	21	396.13	811.40	4646.7	716.64	1597.70	4817.0
20	82.0	6	21	391.21	803.00	4604.9	709.93	1578.60	4733.6
20	79.4	8	21	384.58	788.50	4471.6	699.92	1548.10	4573.4
20	79.1	10	21	379.06	771.70	4373.3	689.10	1508.30	4490.0
20	76.3	12	21	355.79	707.00	3895.0	644.60	1386.20	4015.2
20	74.1	14	21	332.04	642.60	3483.5	599.23	1263.10	3575.0
20	74.0	16	21	302.71	558.70	2978.3	540.10	1092.30	3071.7
20	71.0	18	21	260.97	464.90	2321.9	459.36	900.60	2400.0
20	72.3	20	21	209.50	363.50	1616.7	353.04	692.30	1661.5

These tables suggest show that the coefficient sign variation method is also superior than the other algorithms when applied to polynomials with varying number of

Table 18: **IPIISS** Search Tree Statistics for Polynomials with Random Roots

n	K	r	H	N	L	IPIISS	IPIISSRS
20	88.4	0	1.00	2.00	2.00	78.3	1303.3
20	86.8	2	2.40	4.80	3.40	134.8	1826.6
20	84.6	4	6.80	15.60	8.80	341.8	4131.6
20	82.0	6	7.10	22.00	12.00	463.4	5380.2
20	79.4	8	7.50	26.60	14.30	541.5	6180.0
20	79.1	10	7.80	32.00	17.00	636.6	7025.2
20	76.3	12	8.30	36.80	19.40	701.8	7028.4
20	74.1	14	8.10	42.00	22.00	763.5	6891.4
20	74.0	16	8.10	45.40	23.70	778.1	6096.7
20	71.0	18	8.30	49.60	25.80	783.3	4966.8
20	72.3	20	8.30	53.00	27.50	749.9	3446.7

Table 19: **IPRICS** Search Tree Statistics for Polynomials with Random Roots

n	K	r	H	N	L	IPRICS	T_1	IPRICSD	T_2
20	88.4	0	4.80	10.80	6.40	86.6	110.22	101.9	117.31
20	86.8	2	5.20	13.20	7.60	103.3	107.27	126.6	114.54
20	84.6	4	6.50	18.80	10.40	146.5	101.85	191.6	110.30
20	82.0	6	6.60	21.20	11.60	165.0	95.57	205.5	104.47
20	79.4	8	6.80	25.00	13.50	190.1	91.03	246.7	100.48
20	79.1	10	7.10	26.80	14.40	198.4	88.24	250.0	97.61
20	76.3	12	7.30	31.00	16.50	225.2	85.01	286.6	94.38
20	74.1	14	7.30	34.40	18.20	244.8	82.03	298.3	91.49
20	74.0	16	7.50	35.20	18.60	248.3	79.94	305.0	89.18
20	71.0	18	7.40	37.80	19.90	263.4	77.45	321.5	86.62
20	72.3	20	7.40	42.00	22.00	288.1	77.13	361.7	86.44

Table 20: Comparison of Derivative Sequence Algorithms for Polynomials with Random Roots

n	K	r	D	IPRIDS	D'	IPRIPDS	IPRIDSF	IPRIDTCF
20	88.4	0	75.06	999.9	50.84	876.8	694.5	616.7
20	86.8	2	74.31	1242.0	50.52	1115.1	814.8	718.4
20	84.6	4	73.50	1486.6	50.26	1361.8	891.7	794.9
20	82.0	6	72.94	1641.7	49.2	1475.1	898.4	818.5
20	79.4	8	71.76	1924.9	48.22	1720.1	991.8	933.4
20	79.1	10	71.47	2194.7	47.75	1963.7	1046.7	978.2
20	76.3	12	70.59	2530.0	46.88	2276.9	1129.9	1111.7
20	74.1	14	70.30	2798.1	46.78	2505.2	1203.2	1145.0
20	74.0	16	70.30	2953.4	46.71	2619.6	1235.1	1198.5
20	71.0	18	69.24	3228.4	45.52	2915.0	1321.6	1306.6
20	72.3	20	69.87	3571.8	45.94	3208.3	1401.7	1380.0

Table 21: IPRIDSF Statistics for Random Polynomials

n	k	r	r'	R_0	B	B_l	B_∞	V_0	V_1	V_2
20	88.4	0	1.76	3.5	2.39	2.96	6.20	4.40	16.30	12.70
20	86.8	2	2.47	4.5	2.31	3.55	7.20	6.40	25.80	12.80
20	84.6	4	3.20	6.0	2.24	3.86	6.50	6.60	37.80	12.40
20	82.0	6	3.66	6.9	2.15	3.96	6.70	7.20	45.20	11.20
20	79.4	8	4.67	8.8	1.93	3.96	6.00	10.90	58.80	11.10
20	79.1	10	5.59	10.5	1.73	3.75	5.90	14.00	72.60	9.60
20	76.3	12	6.87	12.4	1.57	3.80	5.70	21.2	89.1	8.30
20	74.1	14	7.94	14.2	1.36	3.70	6.00	32.00	98.40	6.40
20	74.0	16	8.60	16.0	1.27	3.76	5.50	44.50	96.90	6.00
20	71.0	18	9.74	18.0	1.25	3.83	5.40	46.80	116.30	3.90
20	72.3	20	11.05	20.0	1.10	3.58	5.50	64.9	123.7	1.4

random real roots. However, in this case the performance of the Sturm sequence algorithm improves as the number of real roots increases. This is due to the decreasing size of the coefficients of the polynomials in the Sturm sequence. We do not know why the size of the coefficients in the Sturm sequence should decrease as the number of real roots increases. Also, the cost of the derivative sequence algorithm significantly increases as the number of real roots increases. This may be due to the fact that between any two real roots there is a real root of the derivative. Thus as the number of real roots increases, so do the number of real roots of the derivatives. Eventually the cost of the derivative sequence algorithm approaches the cost of the Sturm sequence algorithm.

Polynomials with all Real Roots

Polynomials with all real roots have special properties that can influence the behavior of real root isolation algorithms. For example, both Fourier's theorem and Descartes' rule of signs give the exact number of real roots in an interval when the polynomial has all real roots.

Theorem 59 (Fourier's Theorem with all Real Roots) *Let $A(x)$ be a real polynomial of degree m with all real roots, let $a < b$ be real numbers, and let V_x be the number of variations of the derivative sequence evaluated at x . Then $V_a - V_b$ is equal to the number of real roots in the interval $(a, b]$.*

PROOF. Fourier's Theorem (Theorem 23) implies that $V_{-\infty} - V_{\infty} = m$, and $V_{-\infty} - V_{\infty} = (V_{-\infty} - V_a) + (V_a - V_b) + (V_b - V_{\infty})$, which is equal to $r_1 + 2h_1 + r_2 + 2h_2 + r_3 + 2h_3$,

where r_1 is the number of roots in $(-\infty, a]$, r_2 is the number of roots in $(a, b]$, and r_3 is the number of roots in $(b, \infty]$. Since $r_1 + r_2 + r_3 = m$, the theorem is proved ■

This implies that the tree associated with the coefficient sign variation method will be the same as the tree associated with the Sturm sequence algorithm. Furthermore, the non-algorithm we proposed which uses Fourier's theorem is an algorithm in this case.

Two other important properties of polynomials with all real roots, relate to Sturm sequences and derivative sequences. First, any Sturm sequence must be normal, since $V_{-\infty} - V_{\infty} = m$ and if the Sturm sequence was not normal there could not be m variations. Second, between any two roots of the k -th derivative there is exactly one root of the $k + 1$ -st derivative. Therefore, the total number of real roots of the derivative sequence is $\binom{m}{2}$.

The number of real roots can be determined without actually isolating them using Sturm's theorem. Since the number of real roots is equal to $V_{-\infty} - V_{\infty}$, only the leading coefficients of the Sturm sequence need to be computed. The leading coefficients of the subresultant Sturm sequence can be computed by the modular algorithm in Section 4.1.1 in time dominated by $m^3 L(d)^2$ since only the $m + 1$ leading coefficients need to be lifted. Using this algorithm a polynomial can be checked to determine if it has all real roots.

Several important classes of polynomials have all real roots. Since the eigenvalues of a real symmetric matrix are real, the corresponding characteristic polynomial has all real roots. Another class of polynomials that have all real roots are orthogonal

polynomials. Orthogonal polynomials have the further property that the size of the coefficients in their Sturm sequence are small and therefore the Sturm sequence algorithm behaves well on these polynomials. This was first empirically observed by Collins [15] for the Legendre and Chebycheff polynomials. Properties of orthogonal polynomials can be used to explain this observation.

A sequence of polynomials $f_n(x)$, with $\deg(f_n(x)) = n$, is called orthogonal on the interval $[a, b]$ with respect to the weight function $w(x) > 0$, if

$$\int_a^b f_m(x)f_n(x)w(x)dx = 0 \quad \text{for } m \neq n$$

Orthogonal polynomials have many interesting properties. For an overview see [1] and detailed exposition see [46]. The properties of interest to us are: (1) Orthogonal polynomials satisfy a recurrence relation of the form $f_{n+1}(x) = (a_n + b_n x)f_n(x) - c_n f_{n-1}(x)$, where a_n , b_n , and c_n are constants that can be computed, (2) The derivatives of a sequence of orthogonal polynomials are orthogonal, (3) All of the roots of an orthogonal polynomial are real, simple, and located in the interior of the interval of orthogonality, (4) The roots of $f_{n-1}(x)$ separate the roots of $f_n(x)$.

These properties can be used to show that the primitive Sturm sequence of an orthogonal polynomial is the primitive derivative sequence. Moreover, since the derivatives satisfy a recurrence relation, the size of the coefficients in the derivative sequence typically decrease instead of increasing.

Theorem 60 (Sturm Sequence of an Orthogonal Polynomial) *If $f_n(x)$ is the n -th element in a sequence of orthogonal polynomials, then the primitive Sturm sequence of $f_n(x)$ is, up to sign, the primitive derivative sequence.*

PROOF. Since the sequence of polynomials, $f_n(x)$ is orthogonal, $f_{n+1}(x) = (a_nx + b_n)f_n(x) - c_nf_{n-1}(x)$. Taking the derivative of both sides of this equation shows that $f'_{n+1}(x) = (a_nx + b_n)f'_n(x) + a_nf_n(x) - c_nf'_{n-1}(x)$. Since the sequence of derivatives, $f'_n(x)$ is also orthogonal, $f'_{n+1}(x) = (a'_nx + b'_n)f'_n(x) - c'_nf'_{n-1}(x)$ for some constants a'_n , b'_n , and c'_n . Combining the two equations for f'_{n+1} leads to the equation $a_nf_n(x) = ((a'_n - a_n)x + (b'_n - b_n))f'_n(x) + (c_n - c'_n)f'_{n-1}(x)$. This equation and the recurrence relation for the derivative sequence implies that the primitive remainder sequence of $f_n(x)$ and $f'_n(x)$ is the primitive derivative sequence. Therefore, up to sign, the primitive Sturm sequence of $f_n(x)$ is the primitive derivative sequence of $f_n(x)$ ■

Using this theorem and the properties listed above, it is possible to estimate the computing time for the different root isolation algorithms for orthogonal polynomials. We will carry out this analysis for the Chebycheff polynomials.

The Chebycheff polynomials of the first kind are defined by $T_n(x) = \cos(n\theta)$, where $x = \cos(\theta)$, and the Chebycheff polynomials of the second kind are defined by $U_n(x) = \sin(n\theta)/\sin(\theta)$. Using these definitions it is easy to verify that $T'_n(x) = nU_{n-1}(x)$. The Chebycheff polynomials satisfy the recurrence relation $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$, where $T_1(x) = x$ and $T_0(x) = 1$, and $U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x)$, where $U_1(x) = 2x$ and $U_0(x) = 1$.

Since $T'_n(x) = nU_{n-1}(x)$, it follows from Theorem 60 that the primitive Sturm sequence of $T_n(x)$ is $T_n(x)$ followed by the sequence of Chebycheff polynomials of the second kind $U_{n-1}(x), \dots, U_0(x)$. In particular, the following equation holds:

$$n(n-1)T_n(x) = (n-1)xT'_n(x) - nT'_{n-1}(x).$$

This observation along with an estimate on the separation of the roots of $T_n(x)$ can be used to obtain a codominance relationship, as a function of n , for the computing time of the Sturm sequence algorithm and the coefficient sign variation method for isolating the real roots of $T_n(x)$. This estimate will show that the time for computing the Sturm sequence is less costly than the time required for evaluating the Sturm sequence at the midpoints used during the isolation phase. Even though the Sturm sequence is particularly simple for the Chebycheff polynomials, the computing time function for the coefficient sign variation method is strictly dominated by the computing time function for the Sturm sequence algorithm. The empirical computing times in Table 22 shows that the Sturm sequence algorithm is faster for small n ; however, when $n > 32$, the coefficient sign variation method is faster. Table 22 also lists the computing time for the derivative sequence based algorithm **IPRIDSF**. The computing times for the derivative sequence algorithms are considerably more expensive than the other two algorithms; however, the derivative sequence algorithms isolate all of the roots in the derivative sequence, which implies that as a side effect from isolating the roots of $T_n(x)$ the roots of $U_k(x)$ are also isolated for $k < n$.

Theorem 61 *The computing time for **IPRIST**($T_n(x)$) is codominant with $n^4 \log(n)^3$, and the computing time for **IPRICS**($T_n(x)$) is codominant with $n^4 \log(n)^2$.*

PROOF. The max norm of $T_n(x)$ is codominant with 2^n , and since the Sturm sequence is the Chebycheff polynomials of the second kind, all of the polynomials in the Sturm sequence have max norms dominated by 2^n . Using these bounds it is easy to show that the time for computing the Sturm sequence is dominated by n^4 .

The height of the search tree can be estimated using an estimate of the root separation. The roots of $T_n(x)$ are $\cos((2k+1)\pi/(2n))$ for $k = 0, 1, \dots, n-1$. The Taylor series expansion for $\cos(\theta)$ shows that $(\theta_2^2 - \theta_1^2)/2 < \cos(\theta_1) - \cos(\theta_2) < \theta_2^2/2$. Since the two closest roots are obtained when $k = 0$ and $k = 1$, $\pi^2/n^2 < \text{sep}(T_n(x)) < 9/8\pi^2/n^2$ and $1/\text{sep}(T_n(x))$ is codominant with n^2 and the height of the tree is codominant with $\log(n)$.

The time for the Sturm sequence evaluations at a level codominant with the height of the tree is $n^3 \log(n)^2$. Similarly, the time for performing a translation at a node with level codominant with the height of the tree is $n^3 \log(n)$. Since the number of nodes in the tree is codominant with $n \log(n)$ the theorem is proved ■

Table 22: Timings for Chebycheff Polynomials

n	IPNPRS	IPISS	IPRICS	H	N	L	IPRDSF
8	0	17	33	6	22	12	66
16	33	84	133	8	38	20	433
32	200	1117	867	11	74	38	4833
64	1383	14349	8017	13	138	70	62188

CHAPTER V

Real Algebraic Number Algorithms

This chapter discusses algorithms for performing operations with real algebraic numbers and polynomials with real algebraic number coefficients. The algorithms discussed will be used as subalgorithms for real root isolation of polynomials with real algebraic number coefficients. In Section 5.1, we briefly discuss several ways of representing real algebraic numbers. This section includes a discussion of multiple extensions and interval arithmetic. Section 5.2 presents algorithms for evaluating a polynomial with algebraic number coefficients at a rational number and for performing polynomial transformations on polynomials with algebraic number coefficients. Section 5.3 derives and analyzes several algorithms for computing the sign of an element of a real algebraic number field. Section 5.4 discusses several important properties of the norm of a polynomial with algebraic number coefficients. These properties will be used in the design and analysis of algorithms for isolating the real roots of polynomials with real algebraic number coefficients. Section 5.5 shows how to compute a root bound for a polynomial with real algebraic number coefficients. Section 5.6 discusses results due to Langemyr on computing the gcd of a polynomial with algebraic number coefficients. Algorithms for computing the gcd which use a PRS can be modified to compute a Sturm sequence.

5.1 Representation of Real Algebraic Numbers

5.1.1 Minimal Polynomial and Isolating Interval

A real algebraic number is a real number which satisfies an integral polynomial equation. A real algebraic number α can be represented by a squarefree polynomial $A(x)$ such that $A(\alpha) = 0$ and an isolating interval, I , which distinguishes α from its real conjugates. The assumption that $A(x)$ is squarefree is required by the algorithms that are used for performing arithmetic and order operations in the field obtained by adjoining α to the field of rational numbers. For example $\sqrt{2}$ can be represented with the polynomial $x^2 - 2$ and the isolating interval $(1, 2)$.

The extension field of the rationals generated by α is denoted by $Q(\alpha)$. An element, β , in $Q(\alpha)$ is represented by the polynomial $B(x)$ in the residue class ring $Q[x]/(A(x))$ such that $\beta = B(\alpha)$. If the degree of $A(x)$ is equal to m , $1, x, \dots, x^{m-1}$ is a basis for $Q[x]/(A(x))$ and we can assume that the degree of $B(x)$ is less than m . In general this representation is not unique since we did not require $A(x)$ to be irreducible. If $A(x)$ is irreducible then $Q(\alpha) \cong Q[x]/(A(x))$. Since it is only assumed that $A(x)$ is squarefree, $Q[x]/(A(x)) \cong Q[x]/(A_1(x)) \times \dots \times Q[x]/(A_t(x))$, where $A_1(x), \dots, A_t(x)$ are the irreducible factors of $A(x)$.

Algorithms for performing arithmetic and order computations using this representation were first reported in Rubald's thesis [42]. Arithmetic is performed using polynomial operations in $Q[x]/(A(x))$. Order computation will be discussed in Section 5.3.

An element of $Q(\alpha)$ is represented by a rational polynomial $B(x)$. In order to

minimize the number of rational operations, we will represent $B(\alpha)$ by a primitive integral polynomial $\overline{B}(x)$ and a rational number b such that $B(x) = b\overline{B}(x)$. Experiments have shown that multiplication and inversion are considerably faster using this modified representation, while addition is somewhat slower. For real root isolation of polynomials with coefficients in $Q(\alpha)$, we can assume that the coefficients lie in the subring $Z[\alpha]$. This assumption can be satisfied by multiplying through by the least common multiple of the denominators of the coefficients,

If α is an algebraic number such that $A(\alpha) = 0$, then $a_m\alpha$ is an algebraic integer since $\overline{A}(x) = a_m^m A(x/a_m)$ is a monic integral polynomial and $\overline{A}(a_m\alpha) = 0$. Therefore the field $Q(\alpha)$ is generated by the algebraic integer $a_m\alpha$, and the field $Q(\alpha)$ can be represented with a monic polynomial. The benefit of doing this is that all reduction operations modulo the defining polynomial can be performed with polynomial division and if the coefficients of the polynomial being reduced are integral, they remain integral.

Given an element $\beta \in Q(\alpha)$ it is possible to construct a defining polynomial and isolating interval for β . The algorithm **ANFAF**, listed in Figure 33, produces a defining polynomial and an isolating interval for $\beta = B(\alpha)$. The algorithm uses Theorem 15 to compute a defining polynomial and can use any real root isolation algorithm to find an isolating interval for β .

5.1.2 Multiple Extensions

In this section we show how to extend the representation of a simple algebraic extension field to a multiple extension field. By the primitive element theorem a multiple

ANFAF($A(x), I, B(x); M(y), J$)

[Algebraic number from algebraic field element. Inputs: $A(x)$ a squarefree integral polynomial defining the algebraic number α . I a binary rational isolating interval for α . $B(x)$ an integral polynomial, which represents $\beta = B(\alpha)$ an element of $Z(\alpha)$. Outputs: $M(x)$, defining polynomial for β and J , an isolating interval for β .]

1. [Compute defining polynomial.] $M(y) \leftarrow \text{gsfd}(\text{res}_x(A(x), y - B(x)))$;
2. [Find Isolating Interval.] $L \leftarrow \text{RootIsolation}(M(y))$; $L' \leftarrow L$; repeat {
 $\text{ADV}(L'; J, L')$; $\text{FIRST2}(J; a, b)$; if $a < B(\alpha) < b$ then return } ■

Figure 33: **ANFAF** Algebraic Number from Algebraic Field Element

extension can always be represented by a simple extension. However, the defining polynomial obtained with a constructive version of the primitive element theorem can have large coefficients. Several people have suggested using multiple extensions rather than using the corresponding primitive extensions.

The multiple extension $Q(\alpha_1, \dots, \alpha_r)$ can be viewed as a simple extension of the field $Q(\alpha_1, \dots, \alpha_{r-1})$ provided α_r is defined by a polynomial with coefficients in $Q(\alpha_1, \dots, \alpha_{r-1})$. This leads to a tower of fields and a recursive representation of $Q(\alpha_1, \dots, \alpha_r)$. Let $E_i = Q(\alpha_1, \dots, \alpha_i)$, then $E_i = E_{i-1}(\alpha_i)$ and is represented by a defining polynomial $A_i(\alpha_1, \dots, \alpha_{i-1}, x_i)$ and an isolating interval I_i for α_i . We assume that each $A_i(x_i)$ is squarefree. Finding isolating intervals requires a root isolation algorithm for polynomials with algebraic number coefficients. An element $B(\alpha_1, \dots, \alpha_r)$ of $Q(\alpha_1, \dots, \alpha_r)$ is represented by the r -variate rational polynomial $B(x_1, \dots, x_r)$ with the degree in the i -th variable less than m_i where m_i is the degree

of $A_i(x_i)$. Algorithms for performing arithmetic in $Q(\alpha_1, \dots, \alpha_r)$ with this representation have been presented by Langemyr in [31]. An algorithm for computing the sign of a real algebraic number in the multiple extension $Q(\alpha_1, \dots, \alpha_r)$ is presented in Section 5.3.1.

5.1.3 Using the Derivative Sequence

Coste and Roy have presented an alternative means of distinguishing the real conjugates of an algebraic number which does not rely on isolating intervals [19]. The conjugates of a real algebraic number, α , defined by $A(x)$, can be distinguished by the sequence of signs of $A(\alpha), A^{(1)}(\alpha), \dots, A^{(m)}(\alpha)$. Coste and Roy show that this representation is well defined (the sign sequences for distinct conjugates are different) using Thom's Lemma. Fourier's theorem can be used to give an alternative proof of the correctness of this representation and leads to a stronger result.

Theorem 62 (Coste-Roy) *Let $\alpha_1, \dots, \alpha_r$ be the distinct real roots of $A(x)$, and let $V_x = \text{var}(\text{sign}(A(x)), \dots, \text{sign}(A^{(k)}(x)))$, where k is the smallest integer such that $A^{(k)}(x)$ does not have any real roots. Then $V_{\alpha_1} < \dots < V_{\alpha_r}$. In particular, $V_{\alpha_i} \neq V_{\alpha_j}$ for $i \neq j$ and the sign sequences of the derivative sequence evaluated at α_i and α_j are different.*

PROOF. By Fourier's theorem (Theorem 23, $V_{\alpha_i} - V_{\alpha_j} > 0$ for $i < j$ since there is at least one real root in the interval $(\alpha_i, \alpha_j]$ ■

Coste and Roy show how to compute the different sign sequences of a sequence of polynomials evaluated at the real roots of a polynomial $A(x)$. Their technique relies

on a generalization of Sturm's theorem and an idea from [3] for computing consistent inequalities. Their algorithm does not require isolating intervals for the real roots of $A(x)$. We will not consider this representation in this thesis; however, we note that the derivative sequence algorithms for real root isolation can be used to compute the signs of each of the derivatives at the the real roots of $A(x)$. The derivative sequence algorithm produces isolating intervals for the real roots, along with multiplicities, of each of the derivatives of $A(x)$. If these intervals are refined so that none of them intersect it is possible to compute the sign of $A^{(k)}(\alpha)$, where $A(\alpha) = 0$ by counting the number of roots of $A^{(k)}(x)$ less than α . If there are t roots, with multiplicities $m_1, \dots, m_t$, of $A^{(k)}(x)$ less than α then

$$\text{sign}(A^{(k)}(\alpha)) = (-1)^{\sum_{i=1}^t m_i} \text{sign}(A^{(k)}(-\infty)).$$

Our results comparing the computing times of the Sturm sequence algorithm and the derivative sequence algorithm suggests that this is a faster method, in practice, than the method proposed by Coste and Roy. However, since our method relies on isolating intervals, we might as well use the isolating interval representation. In Section 5.3 we also present an algorithm, **AFSIGNDS**, which computes the signs of $B(\alpha), \dots, B^{(k)}(\alpha)$ for a particular α represented with an isolating interval.

5.1.4 Interval Arithmetic

For our purposes it is not always necessary to have an exact representation for a real algebraic number or an element of a real algebraic number field. If $\alpha \in I$ and $\beta \in J$ are real algebraic numbers, it is possible to compute an interval $I + J$ which

contains $\alpha + \beta$ and an interval IJ which contains $\alpha\beta$. Inductively, it is possible to compute an interval containing any polynomial function of α and β . In many cases, the resulting interval can be used to obtain the desired information about the algebraic numbers. For example, interval arithmetic can be used to compute the sign of $\beta = B(\alpha)$ provided the interval, J , containing β does not contain zero. In Section 5.3, we present an algorithm, which uses interval arithmetic, for computing the signs of elements in $Q(\alpha)$. Section 6.6 discusses methods, using interval arithmetic, for isolating the real roots of a polynomial with coefficients in $Q(\alpha)$. These methods use interval arithmetic to perform arithmetic and sign computations in $Q(\alpha)$.

Interval arithmetic is a formalism for computing with sets of real numbers given by inequalities. If $I = [a, b] = \{x \mid a \leq x \leq b\}$ and $J = [c, d] = \{x \mid c \leq x \leq d\}$, then $I + J = [a + c, b + d] = \{x + y \mid a \leq x \leq b \text{ and } c \leq y \leq d\}$, $I - J = [a - d, b - c] = \{x - y \mid a \leq x \leq b \text{ and } c \leq y \leq d\}$, and $IJ = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] = \{xy \mid a \leq x \leq b \text{ and } c \leq y \leq d\}$.

By checking signs of the endpoints the definition of interval product can be reduced to nine cases, eight of which only require two multiplications and the remaining case requiring four multiplications. The algorithm **BRISUM** and **BRIPROD** implement interval addition and product for binary rational intervals using the special algorithms for binary rational arithmetic discussed in Section 3.1. The book by Moore [41] should be consulted for a more thorough discussion of interval arithmetic.

An interval function $F(X)$, (i.e. a function mapping intervals to intervals) is an *extension* of a real function, $f(x)$ if $F[x, x] = f(x)$. In other words an interval

extension of $f(x)$ is an interval valued function which has real values which coincide with $f(x)$ for real inputs. An interval extension for which $F(X) \subset F(Y)$ for $X \subset Y$ is called inclusion monotonic. These types of interval functions can be used to estimate the range of a function on an interval.

Theorem 63 (Moore) *If $F(X)$ is an inclusion monotonic interval extension of $f(x)$ then $F(X)$ contains the range of $f(x)$ for $x \in X$.*

PROOF. Theorem 3.1 in [41] ■

If a polynomial, $A(x)$, is evaluated at an interval, using interval arithmetic, the resulting interval contains the range of $A(x)$. This easily follows by induction on the number of arithmetic operations in the scheme for evaluating the polynomial. Therefore we can use interval arithmetic to bound the range of a polynomial on an interval. In particular, an interval containing an element $B(\alpha) \in Q(\alpha)$ can be obtained by evaluating $B(x)$ at an isolating interval for α . Moreover, the width of the interval containing $B(\alpha)$ can be made arbitrarily small by decreasing the width, W , of the isolating interval for α .

Theorem 64 (Moore) *If $F(X)$ is an interval extension of $f(x)$ then $W(F(X)) \leq cW(X)$ for all intervals, X , contained in some initial interval X_0 . c is a constant which depends on the initial interval X_0 , the function F , and the scheme for evaluating F .*

PROOF. Lemma 4.1 in [41]. For polynomials it is easy to prove this theorem using induction on the number of arithmetic operations in the scheme for evaluating the polynomial ■

Different methods for evaluating a polynomial produce intervals of different widths and sizes, with some closer to the actual range than others. Using induction and the subdistributivity property of interval arithmetic (i.e. $I(J + K) \subset IJ + IK$), it is possible to show that Horner's method always produces an interval whose width is smaller than or equal to the width of the interval obtained directly by evaluating $A(I) = \sum_{i=0}^m a_i I^i$. For example, let $A(x) = x^2 - 3x + 1$. Then, using this formula, $A([0, 1]) = [0, 1]^2 - 3[0, 1] + 1 = [-2, 2]$. If Horner's method is used, a smaller interval is obtained: $[0, 1]([0, 1] - 3) + 1 = [-2, 1]$. Since the actual range is $[-1, 1]$ both algorithms provide an overestimate.

In order to analyze algorithms which use Horner's method to bound the range of a polynomial, it is necessary to obtain an estimate for the constant in Theorem 64.

Theorem 65 *Let $A(x) = \sum_{i=0}^m a_i x^i$, $I = [a, b]$, $|A(x)|_1 = d$, and $|I| = \max(|a|, |b|)$. Assume $J = A(I)$ is computed using Horner's method. If $|I| > 1$, then $W(J) \leq md|I|^m W(I)$, and if $|I| \leq 1$, then $W(J) \leq mdW(I)$.*

PROOF. The proof is by induction and uses the following property ([41]) of interval arithmetic $W(IJ) \leq W(I)|J| + |I|W(J)$. Let $I_k = I_{k-1}I + a_{m-k}$, be the interval obtained after the k -th step of Horner's method. Then $W(I_k) = W(I_{k-1}I)$, which is less than or equal to $W(I_{k-1})|I| + |I_{k-1}|W(I)$. Furthermore, $|I_k| \leq \sum_{i=0}^k |a_{m-i}||I|^i$, which if $|I| > 1$ is less than or equal to $d|I|^k$ and if $|I| \leq 1$ is less than or equal to d . Therefore, by induction, $W(I_k)$ is less than or equal to $((k-1)d|I|^k + d|I|^{k-1})W(I) \leq kd|I|^k W(I)$ if $|I| > 1$ and less than or equal to $kdW(I)$ if $|I| \leq 1$ ■

Another method for bounding the range of a polynomial, which can be better

than Horner's method (especially if the interval is small), is based on the mean value theorem.

Theorem 66 (Alefeld-Herzberger) *Let c be in the interval $I = [a, b]$. Let $A(x)$ be a polynomial. Then $J = A(c) + A'(I)(I - c)$ contains the range of $A(x)$ for $x \in I$. Moreover, $W(J) \leq c_1 W(I)^2$ for some constant c_1 .*

PROOF. Containment follows from the mean value theorem. By Theorem 64, $W(A'(I)) \leq c_0 W(I)$, and hence $W(A(c) + A'(I)(I - c)) \leq c_1 W(I)^2$ for some constants c_0 and c_1 ■

Another benefit of this approach is that an estimate of the range of the derivative is computed. If zero is not contained in the estimate of the range of the derivative then $A(x)$ is monotonic on the interval I and the range can be computed exactly by evaluating $A(x)$ at the endpoints of I .

The example in Figure 34 compares these three approaches to estimating the range of a polynomial at an interval: (1) Horner's method, (2) the mean value theorem approach, and (3) the mean value theorem approach which checks for monotonicity. The mean value theorem approach uses Horner's method to evaluate the derivative and chooses the point c by using the algorithm **RIB**.

This example is used to estimate the real algebraic number $B(\alpha)$ where $B(x) = 914x^4 - 993x^3 + 3x^2 - 505x + 415$ and α is the root of $A(x) = -916x^5 + 592x^4 - 243x^3 - 788x^2 - 81x + 127$ in the interval $I = [0, 2]$. The interval computed by Horner's method is J and the interval for the derivative, computed by Horner's method is J' . The interval computed using the mean value theorem is MVT, and if J' does not

contain 0 the exact range is computed. After the bounds for the range of $B(I)$ are computed, I is bisected and the bounds are computed for the subinterval containing α .

Initially, the bound obtained using Horner's method is better than the bound using the mean value theorem; however, after six steps, MVT is closer to the actual range and MVT is seen to converge faster. However, the size of the endpoints seem to be larger for MVT than for straightforward evaluation using Horner's rule.

The algorithm **IUPBRIEV**($A(x), I$) evaluates a polynomial $A(x)$ at a binary rational interval, I , using Horner's method. The algorithm **IUPBRIMVEV**($A(x), I$) evaluates a polynomial $A(x)$ at a binary rational interval using the mean value theorem approach. If **IUPBRIMVEV** can determine that $A(x)$ is monotonic on I , then the exact range is returned. Table 23 lists the computing times, in milliseconds, for these two algorithms, when applied to random polynomials and random standard intervals whose endpoints have k bits in the numerator and denominator. The times are in milliseconds and are averaged over ten iterations.

5.2 Algebraic Polynomial Operations

In this section we present algorithms for polynomial evaluation and transformations. These algorithms apply to multivariate integral polynomials. Since we can view a polynomial in $Z[\alpha][y]$ as a bivariate integral polynomial, these algorithms apply to polynomials with algebraic number coefficients. For real root isolation, we can always assume that a polynomial with coefficients in $Q(\alpha)$ has coefficients in $Z[\alpha]$ since we can clear the denominators. Some of the algorithms in this section need to permute

$I = (0, 2)$	
$J = (-8527, 6097)$	Length = 14624.00
$J' = (-12409, 16839)$	Length = 29248.00
$MVT = (-17005, 16673)$	Length = 33678.00
$I = (0, 1)$	
$J = (-1080, 415)$	Length = 1495.00
$J' = (-3478, 178)$	Length = 3656.00
$MVT = (-6571/4, 7341/4)$	Length = 3478.00
$I = (0, 1/2)$	
$J = (313/8, 415)$	Length = 375.88
$J' = (-4987/4, -502)$	Length = 744.75
$MVT = (-4441/128, 75351/128)$	Length = 623.38
$Range = (385/4, 415)$	Length = 318.75
$I = (1/4, 1/2)$	
$J = (1083/16, 4489/16)$	Length = 212.88
$J' = (-4073/4, -9207/16)$	Length = 442.81
$MVT = (132045/2048, 653389/2048)$	Length = 254.56
$Range = (385/4, 35455/128)$	Length = 180.74
$I = (1/4, 3/8)$	
$J = (190189/1024, 71367/256)$	Length = 93.05
$J' = (-50761/64, -604)$	Length = 189.14
$MVT = (6105393/32768, 9354097/32768)$	Length = 99.14
$Range = (392717/2048, 35455/128)$	Length = 85.24
$I = (5/16, 3/8)$	
$J = (773095/4096, 3893435/16384)$	Length = 48.89
$J' = (-97409/128, -21125/32)$	Length = 100.85
$MVT = (99827913/524288, 124764617/524288)$	Length = 47.56
$Range = (392717/2048, 7729745/32768)$	Length = 44.14
$I = (11/32, 3/8)$	
$J = (1558529/8192, 28226133/131072)$	Length = 25.10
$J' = (-190705/256, -88697/128)$	Length = 52.00
$MVT = (1605749001/8388608, 1801030921/8388608)$	Length = 23.28
$Range = (392717/2048, 112296265/524288)$	Length = 22.43

Figure 34: Interval Evaluation of the Range of a Polynomial

Table 23: Empirical Computing Times (in ms) of Interval Polynomial Evaluation

$\deg(A(x))$	$L(A(x) _\infty)$	k	IUPBRIEV	IUPBRIMVEV
10	10	10	11.7	15.0
10	10	155	48.3	83.3
10	10	300	120.1	198.3
10	155	10	15.0	21.6
10	155	155	55.0	93.3
10	155	300	133.3	218.3
10	300	10	18.3	21.7
10	300	155	60.0	100.0
10	300	300	143.4	235.0
20	10	10	25.0	28.3
20	10	155	170.0	264.8
20	10	300	473.4	694.9
20	155	10	30.0	40.0
20	155	155	181.6	276.8
20	155	300	494.9	738.2
20	300	10	33.3	44.9
20	300	155	198.4	296.8
20	300	300	508.3	751.7
30	10	10	41.7	49.9
30	10	155	375.2	540.0
30	10	300	1054.9	1543.4
30	155	10	48.3	65.0
30	155	155	388.4	561.8
30	155	300	1086.7	1588.5
30	300	10	53.3	73.4
30	300	155	408.3	598.5
30	300	300	1113.4	1629.8

the variables so that the variable being transformed or evaluated at becomes the leading variable. If this is done, appropriate univariate algorithms can be applied to the univariate polynomial coefficients.

5.2.1 Permutation of Variables

In this section we present an efficient algorithm for permuting the variables of a polynomial represented in recursive canonical form. In the recursive representation, a polynomial in r variables is represented as a polynomial in a main variable whose coefficients are polynomials in $r - 1$ variables. The polynomial $A(x_1, \dots, x_r) = \sum_{i=1}^m a_i(x_1, \dots, x_{r-1})x_r^{e_i}$ is represented with the list $(e_1, a_1, \dots, e_m, a_m)$ where $e_1 > \dots > e_m$ are the exponents and $a_1, \dots, a_m$ are the recursive representations of the coefficients. For example $A(x, y) = (3x^3 + 2x - 3)y^3 + (7x^2 + 2)y^2 + 5x^2 + 2x + 1$ is represented by $(3, (3, 3, 1, 2, 0, -3), 2, (2, 7, 0, 2), 0, (2, 5, 1, 2, 0, 1))$.

The current approach in SAC-2 to permuting the variables of a polynomial is to first convert the polynomial to a distributive representation, permute the exponent vectors, sort the terms in the distributive representation, and then convert back to the recursive representation. The algorithm **PPERMV** uses this approach to permute the variables of an r -variate polynomial.

In the distributive representation, the polynomial

$$A(x_1, \dots, x_r) = \sum_{i_1, \dots, i_r} a_{i_1, \dots, i_r} x_1^{e_{i_1}} \dots x_r^{e_{i_r}},$$

is represented with a list $(a_1, E_1, \dots, a_t, E_t)$, where $E_1 < \dots < E_t$ under anti-

lexicographic ordering of the variables. For example,

$$A(x, y) = 3x^3y^3 + 2xy^3 - 3y^3 + 7x^2y^2 + 2y^2 + 5x^2 + 2x + 1$$

is represented by the list

$$(3, (3, 3), 2, (1, 3), -3, (0, 3), 7, (2, 2), 2, (0, 2), 5, (2, 0), 2, (1, 0), 3, (0, 0)).$$

It is easy to convert from recursive to distributive representation by recursively distributing the main variable over the coefficients. Similarly, by recursively collecting terms with the same power in the main variable, it is possible to convert from distributive to recursive representation.

We show how **PPERMV** permutes the variables x and y in $A(x, y)$. After converting $A(x, y)$ to distributive form, the exponent vectors are permuted obtaining:

$$A_1(x, y) = 3y^3x^2 + 2y^3x + 7y^3 + 7yx^3 - 3y - 7x^2 + 3.$$

Next the terms of $A_1(x, y)$ are sorted under anti-lexicographic ordering of the variables (y, x) obtaining:

$$7yx^3 - 3y^3x^2 + 7x^2 + 2y^3x + 7y^3 + 3y - 3.$$

When this is converted back to recursive representation, we obtain the permuted polynomial $(7y)x^3 - (3y^3 + 7)x^2 + (2y^3)x + (7y^3 + 3y - 3)$.

The difficulty with this approach to permuting the variables is that it is too slow. If M is the number of terms in the distributive representation of a polynomial, then the time for **PPERMV** is codominant with $M \log(M)$ if a fast sorting algorithm is used. In fact, the current implementation uses insertion sort so that the time is codominant with M^2 .

Theorem 67 (Computing Time of PPERMV) *Let $\deg_i(A(x_1, \dots, x_r)) = m_i$ and $M = \prod_{i=1}^r (m_i + 1)$. Then the computing time of PPERMV is dominated by $M \log(M) + r \log(r)M$.*

In the remainder of this section we present an algorithm which is linear in M . The idea behind the alternative approach is a fast algorithm for transposing two consecutive variables. Repeated calls to this algorithm are then used to obtain more general permutations.

Suppose

$$A(x, y) = \sum_{j=0}^n a_j(x) y^j = \sum_{j=0}^n \left(\sum_{i=0}^m a_{ij} x^i \right) y^j.$$

If the permuted polynomial $\tilde{A}(y, x) = A(x, y)$, then

$$\tilde{A}(y, x) = \sum_{i=0}^m \tilde{a}_i(y) x^i = \sum_{i=0}^m \left(\sum_{j=0}^n a_{ij} y^j \right) x^i.$$

Therefore, if the coefficients of $A(x, y)$ are stored in a two dimensional array, the coefficients of $\tilde{A}(y, x)$ are obtained by taking the transpose of the matrix containing the coefficients of $A(x, y)$. Since we are using a sparse polynomial representation, the algorithm we use for transposing the variables is similar to sparse matrix transposition. The algorithm PTMV, listed in Figure 35, transposes the main two variables of an r -variate polynomial thought of as a bivariate polynomial with $(r-2)$ -variate polynomial coefficients. To illustrate this algorithm, we trace it on the previous example (see Figure 36). In the trace, $\text{rep}()$ refers to the list representation of a polynomial.

Theorem 68 (Computing Time of PTMV)

Let $\deg_x(A(x, y)) = m$ and $\deg_y(A(x, y)) = n$. Then the computing time of PTMV

$$B(x_1, \dots, x_{r-2}, y, x) \leftarrow \text{PTMV}(r, A(x_1, \dots, x_{r-2}, x, y))$$

[Polynomial Transposition of main variables.]

1. [Transposition.] $\tilde{A}(x, y) \leftarrow A(x, y); B(y, x) \leftarrow 0;$
 while $\tilde{A}(x, y) \neq 0$ do {
 $e \leftarrow \deg_x(\tilde{A}(x, y)); A'(x, y) \leftarrow \tilde{A}(x, y); B_1(y, x) \leftarrow 0;$
 while $A'(x, y) \neq 0$ do {
 $a_1(x) \leftarrow \text{ldcf}(A'(x, y)); f \leftarrow \deg_y(A'(x, y));$
 if $\deg_x(a_1(x)) = e$ then {
 $B_1(y) \leftarrow \text{ldcf}(a_1(x))y^f + B_1(y); \tilde{A}(x, y) \leftarrow \tilde{A}(x, y) - \text{ldcf}(a_1(x))x^e y^f$ }
 $A'(x, y) \leftarrow A'(x, y) - a_1(x)y^f$ };
 $B(y, x) \leftarrow B(y, x) + B_1(y)x^e$ } ■

Figure 35: PTMV Polynomial Transposition of Variables

is dominated by $(m+1)(n+1)$.

The algorithm **PTV** (Polynomial Transposition of Variables) applies the algorithm **PTMV** to the coefficients of an r -variate polynomial to transpose the $(i-1)$ -st and i -th variables. Since there are at most $(m_{i+1}+1) \cdots (m_r+1)$ i -variate coefficients, the following computing time bound is obtained.

Theorem 69 (Computing Time of PTV) *Let $\deg_i(A(x_1, \dots, x_r)) = m_i$. Then the time for **PTV** to transpose the $(k-1)$ -th and k -th variables is dominated by $\prod_{i=k-1}^r (m_i+1)$*

Transposition of consecutive variables can be used to obtain a cyclic permutation. The cyclic permutation $(1, 2, \dots, i)$ can be factored as $(i, i-1)(i, 2, \dots, i-1)$, where multiplication is from left to right. Recursively applying this factorization we obtain

$A^-(x,y) = (3x^3 + 2x - 3)y^3 + (7x^2 + 2)y^2 + (5x^2 + 2x + 1)$
 $\text{rep}(A^-) = (3, (3,3,1,2,0,-3), 2, (2,7,0,2), 0, (2,5,1,2,0,1))$

$B_1(y) = 3y^3$
 $A^-(x,y) = (2x - 3)y^3 + (7x^2 + 2)y^2 + (5x^2 + 2x + 1)$
 $\text{rep}(A^-) = (3, (1,2,0,-3), 2, (2,7,0,2), 0, (2,5,1,2,0,1))$
 $B(y,x) = (3y^3)x^3$
 $\text{rep}(B) = (3, (3,3))$

$B_1(y) = 7y^2 + 5$
 $A^-(x,y) = (2x - 3)y^3 + (2)y^2 + (2x + 1)$
 $\text{rep}(A^-) = (3, (1,2,0,-3), 2, (0,2), 0, (1,2,0,1))$
 $B(y,x) = (3y^3)x^3 + (7y^2 + 5)x^2$
 $\text{rep}(B) = (3, (3,3), 2, (2,7,0,5))$

$B_1(y) = 2y^3 + 2$
 $A^-(x,y) = (-3)y^3 + (2)y^2 + (1)$
 $\text{rep}(A^-) = (3, (0,-3), 2, (0,2), 0, (0,1))$
 $B(y,x) = (3y^3)x^3 + (7y^2 + 5)x^2 + (2y^3 + 2)x$
 $\text{rep}(B) = (3, (3,3), 2, (2,7,0,5), 1, (3,2,0,2))$

$B_1(y) = -3y^3 + 2y^2 + 1$
 $A^-(x,y) = 0$
 $B(y,x) = (3y^3)x^3 + (7y^2 + 5)x^2 + (2y^3 + 2)x + (-3y^3 + 2y^2 + 1)$
 $\text{rep}(B) = (3, (3,3), 2, (2,7,0,5), 1, (3,2,0,2), 0, (3,-3,2,2,0,1))$

Figure 36: Trace of PTMV

$(1, 2, \dots, i) = (i, i-1)(i-1, i-2) \cdots (2, 1)$. If p is a permutation, its action on the variables of a polynomial is defined by $x_i \rightarrow x_{p^{-1}(i)}$. Thus the cyclic permutation $(1, \dots, i)$ permutes the variables $x_1, \dots, x_{i-1}, x_i$ to $x_i, x_1, \dots, x_{i-1}$. The algorithm **PCPV**, listed in Figure 37, uses this factorization and the subalgorithm **PTV** to cyclically permute the variables $x_i, x_{i+1}, \dots, x_j$.

$$B(x_1, \dots, x_j, \dots, x_i, \dots, x_r) \leftarrow \mathbf{PCPV}(r, A(x_1, \dots, x_r), i, j)$$

[Polynomial cyclic permutation of variables. Inputs: $r \geq 2$ is an integer. $A(x_1, \dots, x_r)$ is an r -variate polynomial. $1 \leq i \leq r$ is an integer. $i < j \leq r$ is an integer. Outputs: $B(x_p^{-1}(1), \dots, x_p^{-1}(r)) = A(x_1, \dots, x_r)$, where $p = (i, i+1, \dots, j)$ is a cyclic permutation.]

1. $[(i, \dots, j) = (j, j-1)(i, \dots, j-1).]$ $B_1 \leftarrow \mathbf{PTV}(r, A, j);$
 if $j = i+1$ then
 $B \leftarrow B_1$; return
 else
 $B \leftarrow \mathbf{PCPV}(r, B_1, i, j-1)$ ■

Figure 37: **PCPV** Polynomial Cyclic Permutation of Variables

Theorem 70 (Computing Time of PCPV) Let $\deg_i(A(x_1, \dots, x_r)) = m_i \geq 1$. Then the time for **PCPV** to cyclically permute the variables $x_k, \dots, x_r$ is dominated by $\prod_{i=k}^r (m_i + 1)$

PROOF. Let $S_k = (m_r + 1)(m_{r-1} + 1) + (m_r + 1)(m_{r-1} + 1)(m_{r-2} + 1) + \cdots + (m_r + 1) \cdots (m_k + 1)$. By repeated application of Theorem 69 the computing time for **PCPV** is dominated by S_k . If $m_i > 0$ for $i = k, \dots, r$ then we can inductively show that $S_k \leq 2(m_r + 1) \cdots (m_k + 1)$.

$$S_k = S_{k+1} + (m_r + 1) \cdots (m_k + 1)$$

$$S_k \leq 2(m_r + 1) \cdots (m_{k+1} + 1) + (m_r + 1) \cdots (m_k + 1)$$

$$S_k \leq 2(m_r + 1) \cdots (m_k + 1) \blacksquare$$

A similar factorization can be used for the inverse cyclic permutation $(i, i + 1, \dots, r)^{-1} = (i, i + 1)(i + 1, i + 2) \cdots (r, r - 1)$. The specification for the algorithm **PICPV** is listed in Figure 38.

$$B(x_1, \dots, x_{i+1}, \dots, x_j, x_i, \dots, x_r) \leftarrow \mathbf{PICPV}(r, A(x_1, \dots, x_r), i, j)$$

[Polynomial inverse cyclic permutation of variables. Inputs: $r \geq 2$ is an integer. $A(x_1, \dots, x_r)$ is an r -variate polynomial. $1 \leq i \leq r$ is an integer. $i < j \leq r$ is an integer. Outputs: $B(x_p^{-1}(1), \dots, x_p^{-1}(r)) = A(x_1, \dots, x_r)$, where $p = (i, i + 1, \dots, j)^{-1}$ is an inverse cyclic permutation.]

Figure 38: **PICPV** Polynomial Inverse Cyclic Permutation of Variables

These ideas can be extended to obtain a general permutation; however, since the cyclic permutations are the only ones used in this thesis, we will be content with these results.

Figure 24 compares the computing time of **PCPV** and **PPERMV**. In this experiment, dense polynomials with the same degree in each variable were used, and times were averaged over ten iterations.

5.2.2 Algebraic Polynomial Evaluation

Algebraic polynomial evaluation at a binary rational number can be performed using integer arithmetic. Assuming $A(\alpha, y)$ is in $Z[\alpha, y]$ and has degree m , then $f^m A(x, e/f)$ can be computed by first permuting the variables x and y and then evaluating the univariate polynomial coefficients at e/f . Let $\tilde{A}(y, x) = A(x, y)$ and

Table 24: Empirical Computing Times (in ms) for Permutation of Variables

r	P	$\deg_i(A(x))$	PPERMV	PCPV
2	(2,1)	5	13.4	1.6
2	(2,1)	10	96.7	1.7
2	(2,1)	15	373.4	6.7
2	(2,1)	20	1043.3	11.7
3	(3,1,2)	5	300.0	16.0
3	(3,1,2)	10	9434.0	33.0
3	(3,1,2)	15	88883.0	117.0
3	(3,1,2)	20	459865.0	266.0

$\tilde{A}(y, x) = \sum_{j=0}^n \tilde{a}(y)x^j$. Then $\sum_{j=0}^n f^m(\tilde{a})(e/f)x^j$ is computed using the univariate integral polynomial binary rational evaluation algorithm **IUPBREI**.

Using the cyclic permutation **PCPV** from Section 5.2.1, this same idea can be used to evaluate a multivariate integral polynomial at a specified variable. The variables $x_1, \dots, x_i$ are cyclically permuted so that the i -th variable becomes the leading variable, and then **IUPBREI** is used to evaluate each of the univariate polynomial leading coefficients. The algorithm **IPBREI**, listed in Figure 39, evaluates the i -th variable of an r -variate integral at a binary rational number, $e/2^k$. **IPBREI** returns the integral polynomial $2^{km_i} A(x_1, \dots, x_{i-1}, e, x_{i+1}, \dots, x_r)$, where $m_i = \deg_{x_i}(A(x))$. **IPBREI** uses the subalgorithm **IPBRELVI** (Integral Polynomial Binary Rational Evaluation at the Leading Variable, Integer Part), which in turn uses the univariate algorithm **IUPBREI** to evaluate the univariate leading coefficients.

Theorem 71 (Computing Time of IPBREI) *Assume $A(x_1, \dots, x_r)$ is an inte-*

$$B(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_r) \leftarrow \mathbf{IPBREI}(r, A(x_1, \dots, x_r), i, e, k)$$

[Integral polynomial binary rational evaluation, integral part. Inputs: $r \geq 2$ is an integer. $A(x_1, \dots, x_r)$ is an r -variate integral polynomial. $1 \leq i \leq r$ is an integer. e and k are integers. Outputs: $B(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_r) = 2^{km_i} A(x_1, \dots, x_{i-1}, e, x_{i+1}, \dots, x_r)$, where $m_i = \deg_{x_i}(A(x))$.]

1. [$A = 0$.] if $A = 0$ then { $B \leftarrow 0$; return }.
2. [$i = 1$.] $m \leftarrow \deg_i(A(x_1, \dots, x_r))$;
if $i = 1$ then { $B \leftarrow \mathbf{IPBRELVI}(r, A, e, k, m)$; return }.
3. [$i > 1$.] $\tilde{A} \leftarrow \mathbf{PCPV}(r, A, 1, i)$; $B \leftarrow \mathbf{IPBRELVI}(r, \tilde{A}, e, k, m)$ ■

Figure 39: **IPBREI** Integral Polynomial Binary Rational Evaluation, Integral Part

gral polynomial with $\deg_i(A(x_1, \dots, x_r)) = m_i$, and let $r = e/2^k$ be a binary rational number. The algorithm **IPBREI** computes $2^{km_i} A(x_1, \dots, x_{i-1}, e/2^k, x_{i+1}, \dots, x_r)$ in time dominated by

$$\left\{ \prod_{j \neq i} m_j \right\} \left\{ m_i L(|A(x_1, \dots, x_r)|_\infty) L(e) + m_i^2 (L(e)^2 + L(e)k) \right\}.$$

5.2.3 Algebraic Polynomial Transformations

This section presents algorithms for computing polynomial homothetic transformations, and polynomial translations for polynomials with algebraic number coefficients. The algorithm **PRT** of Section 3.2, can be used for polynomials with algebraic number coefficients since it works independently of the coefficient type. Since the translations and homothetic transformations are integral, the algebraic polynomial transformations can be performed using integral arithmetic assuming the polynomial is in

$Z[\alpha][x]$. This is a fair assumption for root isolation algorithms since the denominators can be cleared. In general, polynomial transformations can be applied to polynomials in $Z[\alpha_1, \dots, \alpha_r][x]$, which can be viewed as integral polynomials in $r + 1$ variables.

The algorithm for binary homothetic transformations is the same as the algorithm for univariate integral polynomials in Section 3.2, except that the coefficients may be integral polynomials. A subalgorithm **IPP2P** (Integral Polynomial Power of 2 Product) is used to multiply an integral polynomial by a power of two. The specification for the algorithm **IPBHT**, which performs a binary homothetic transformation on an integral polynomial, is listed in Figure 40.

$$B(x_1, \dots, x_r) \leftarrow \text{IPBHT}(r, A(x_1, \dots, x_r), i, k)$$

[Integral polynomial binary homothetic transformation. Inputs: $r \geq 1$ is an integer. $A(x_1, \dots, x_r)$ is an r -variate integral polynomial. $1 \leq i \leq r$ is an integer. k is an integer. Outputs: if $k \geq 0$ then $B(x_1, \dots, x_r) = A(x_1, \dots, 2^k x_i, \dots, x_r)$. If $k < 0$ then $B(x_1, \dots, x_r) = 2^{-km_i} A(x_1, \dots, 2^k x_i, \dots, x_r)$, where $m_i = \deg_{x_i}(A(x))$.]

Figure 40: **IPBHT** Integral Polynomial Binary Homothetic Transformation

Theorem 72 (Computing Time of IPBHT) *Assume $A(x_1, \dots, x_r)$ is an integral polynomial with $\deg_i(A(x_1, \dots, x_r)) = m_i$, then **IUPBHT** computes $A(x_1, \dots, 2^k x_i, \dots, x_r)$ in time dominated by*

$$\left\{ \prod_{j \neq i} m_j \right\} \left\{ m_i L(|A(x_1, \dots, x_r)|_\infty) + km_i^2 \right\}.$$

Polynomial translations are performed by permuting the variable to be translated to the leading variable, applying the fast univariate polynomial translation algorithms to the univariate polynomial leading coefficients, and then permuting the translated

variable to its original position. The algorithm **IPTR** (Integral Polynomial Translation) and **IPTR1** (Integral Polynomial Translation by 1) first cyclically permute the variables $x_1, \dots, x_i$ with **PCPV**, and then use **IUPTR** and **IUPTR1** respectively to translate the univariate coefficients. After the translations have been performed, the variables $x_i, x_1, \dots, x_{i-1}$ are permuted back to the original ordering using the algorithm **PICPV**. The specification for **IPTR** is listed in Figure 41. The specification for **IPTR1** is the same except the variable b is not required and the translation is by one.

$$B(x_1, \dots, x_r) \leftarrow \mathbf{IPTR}(r, A(x_1, \dots, x_r), i, b)$$

[Integral polynomial translation. Inputs: $r \geq 1$ is an integer. $A(x_1, \dots, x_r)$ is an r -variate integral polynomial. $1 \leq i \leq r$ is an integer. b is an integer. Outputs: $B(x_1, \dots, x_r) = A(x_1, \dots, x_i + b, \dots, x_r)$.]

Figure 41: **IPTR** Integral Polynomial Translation

Theorem 73 (Computing Time of IPTR and IPTR1) *Assume $A(x_1, \dots, x_r)$ is an integral polynomial with $\deg_i(A(x_1, \dots, x_r)) = m_i$, and b is an integer. **IPTR**, computes $A(x_1, \dots, x_i + b, \dots, x_r)$ in time dominated by*

$$\left\{ \prod_{j \neq i} m_j \right\} \left\{ m_i^3 L(b)^2 + m_i^2 L(b) L(|A(x_1, \dots, x_r)|_\infty)^2 \right\}.$$

IPTR1, computes $A(x_1, \dots, x_i + 1, \dots, x_r)$ in time dominated by

$$\left\{ \prod_{j \neq i} m_j \right\} \left\{ m_i^3 + m_i^2 L(|A(x_1, \dots, x_r)|_\infty)^2 \right\}.$$

5.3 Sign Computation

In Section 4.3 we sketched a general procedure for computing the sign of an element of a real algebraic number field $Q(\alpha)$. If $A(\alpha) = 0$, $\alpha \in I = (a, b]$, and $\beta = B(\alpha)$, the sign of β can be computed by refining the isolating interval, I , until it contains no roots of $B(x)$. If I contains no roots of $B(x)$, the sign of $B(x)$ is invariant for all x in I . In particular, the sign of $\beta = B(\alpha)$ is equal to the sign of $B(x)$ for any x in I . In this section, several algorithms based on this idea are presented and compared. Some algorithms based on this approach are also discussed by Rump in [44]. Alternative approaches to computing the sign of an element of a real algebraic number field are discussed by Kempfert in [28].

Since $A(x)$ is not assumed to be irreducible, we must first check if $B(\alpha) = 0$. Let $C(x) = \gcd(A(x), B(x))$. Then $B(\alpha) = 0$ if and only if $C(\alpha) = 0$. Since any root of $C(x)$ is a root of $A(x)$, and I is an isolating interval for $A(x)$, the only possible root of $C(x)$ in the interval I is α . Moreover, since $A(x)$ is squarefree, $C(x)$ has only simple roots, and $C(\alpha) = 0$ if and only if $C(a)C(b) < 0$ or $C(b) = 0$. This leads to the algorithm **AFZTEST**, listed in Figure 42, for determining if an element of $Q(\alpha)$ is equal to zero.

If $B(\alpha) \neq 0$ then I can be refined so that it contains no roots of $B(x)$. Several algorithms will be used to determine if I contains any roots of $B(x)$. If $B(x)$ does not contain any roots, the sign of $B(\alpha)$ is equal to the sign of $B(b)$, which can be computed since b is rational. If it is not known whether $B(x)$ has a root in I , I is bisected and the subinterval containing α is tested to see whether it contains any

$$s = \mathbf{AFZTEST}(A(x), I, B(x))$$

[Algebraic field element zero test. Inputs: $A(x)$ is a squarefree integral polynomial. I is a binary rational isolating interval for a root, α of $A(x)$. $B(x)$ is an integral polynomial. Outputs: $s \in \{0, 1\}$. $s = 0$ if $B(\alpha) = 0$, otherwise $s = 1$.]

1. [Compute gcd.] $C(x) \leftarrow \gcd(A(x), B(x))$.
2. [Check if $C(\alpha) = 0$.] **FIRST2**($I; a, b$); if $C(a)C(b) < 0 \vee C(b) = 0$ then $s \leftarrow 0$
else $s \leftarrow 1$ ■

Figure 42: **AFZTEST** Algebraic Field Element Zero Test

roots of $B(x)$.

The first algorithm uses the algorithm **DESCARTES** to determine if there are any roots of $B(x)$ in the interval I . The polynomial $B(x)$ is transformed to a polynomial $B^*(x)$ whose positive roots correspond to the roots of $B(x)$ in the interval I . If $\text{var}(B^*(x)) = 0$ then $B(x)$ does not contain any root in the interval I , otherwise I is bisected and the transformed polynomial $B_1^*(x)$, whose positive roots correspond to the roots of $B(x)$ in the subinterval of I containing α , is computed and its coefficient sign variations are counted. This process continues until zero variations are obtained. Since $B(\alpha) \neq 0$, eventually $B^*(x)$ will have no roots with positive real parts and by Theorem 25 $\text{var}(B^*(x))$ will equal zero. The algorithm **AFSIGN**, listed in Figure 43, implements this idea.

Theorem 74 (Computing Time of AFSIGN) Assume $\deg(A(x)) = m$, $\deg(B(x)) = n$ with $m > n$ and $|A(x)|_1 = d$ and $|B(x)|_1 = e$. Let $I_1, \dots, I_r$ be isolating intervals for $\alpha_1, \dots, \alpha_r$ be the distinct real roots of $A(x)$. Then the time to

AFSIGN($A(x), I, B(x); I^*, s$)

[Algebraic field element sign. Inputs: $A(x)$ is a squarefree integral polynomial. I is a standard isolating interval for a real root, α of $A(x)$. $B(x)$ is an integral polynomial such that $B(\alpha) \neq 0$. Outputs: I^* is a standard subinterval of I which does not contain any roots of $B(x)$. $s \in \{-1, 1\}$. $s = \text{sign}(B(\alpha))$.]

1. [α rational] if $\deg(A(x)) = 0 \vee W(I) = 0$ then { $I^* \leftarrow I$; $s \leftarrow \text{sign}(B(b))$.
2. [β rational.] if $\deg(B(x)) = 0$ then { $I^* \leftarrow I$; $s \leftarrow \text{sign}(\text{ldcf}(B(x)))$.
3. [Compute transformed polynomial $\tilde{B}(x)$ whose roots in $(0, 1)$ correspond to the roots of $B(x)$ in I .] $I = (a/2^h, (a+1)/2^h]$; $\tilde{B}(x) = T_{-a}H_{2^h}(B(x))$.
4. [Obtain an isolating interval for alpha containing no roots of $B(x)$ and evaluate the sign of $B(x)$ at the midpoint.] $t \leftarrow \text{sign}(A(b))$; $I^* \leftarrow I$;
 repeat {
 FIRST2($I^*; a^*, b^*$); $c^* \leftarrow \text{RIB}(I^*)$; $B^*(x) \leftarrow T_{-1}R(\tilde{B}(x))$;
 if $\text{var}(B^*(x)) = 0 \wedge x \nmid B^*(x)$ then { $s \leftarrow \text{sign}(B(w^*))$; return };
 $\tilde{B}(x) \leftarrow H_2(\tilde{B}(x))$; $w \leftarrow \text{sign}(A(c^*))$;
 if $tw < 0 \vee t = 0$ then {
 $a^* \leftarrow c^*$; $\tilde{B}(x) \leftarrow \tilde{B}(x+1)$ }
 else {
 $b^* \leftarrow c^*$; $t \leftarrow w$ };
 $I^* \leftarrow (a^*, b^*)$; } ■

Figure 43: **AFSIGN** Algebraic Field Element Sign

compute $\text{sign}(B(\alpha_i))$ for $i = 1, \dots, r$ using **AFSIGN** is dominated by $m^5L(de)^3$.

PROOF. The cost of **AFSIGN** is dominated by the cost of the polynomial translations needed to compute $B^*(x)$ and the cost of evaluating $A(c^*)$. Therefore, to bound the computing time we must bound the cost of these operations and the number of bisections.

A bisection is required if and only if $\text{var}(B^*(x)) > 0$, and by Theorem 25 this can not happen if the circle centered about the interval I^* containing α does not have any roots (real or complex) of $B(x)$. Suppose the refined interval, I^* , is obtained after h bisections. Let k be the number of real conjugates of α that need to be bisected at level h . Then by Davenport's theorem (Theorems 3 and 4) applied to the real and complex conjugate roots of the polynomial $A(x)B(x)$ implies that hk is dominated by $mL(de)$.

The size of the coefficients of the polynomial $B^*(x)$ and the size of the midpoint of I^* are dominated by $mL(de)$. Therefore, by Theorem 35, the cost of the k translations are dominated by $m^3hk + m^2L(e)$ which is dominated by $m^4L(de)$. By Theorem 38, the cost of evaluating $A(x)$ at the midpoint of the k different isolating intervals I^* is dominated by $kmL(d)L(c^*) + km^2L(c^*)^2$ which is dominated by $m^4L(de)^2$.

By Theorem 2, the maximum number of bisections for any conjugate, α_i is dominated by $mL(de)$. Therefore, the total time is dominated by $m^5L(de)^3$ ■

The computing time obtained in this theorem is more than the time to isolate the roots of $A(x)B(x)$, which suggests that **AFSIGN** can be improved (at least asymptotically). Indeed this is the case. The $m^5L(d)^3$ term comes from evaluating

$A(x)$ at the endpoints of the interval I^* . This can be reduced if instead of evaluating $A(x)$ at the endpoints, Descartes' rule is used to select the subinterval of I^* containing α . This is done in the same way that we check to see if $B(x)$ has any roots in I^* . Let $A^*(x)$ be the transformed polynomial whose positive roots correspond to the roots of $A(x)$ in I^* . Let $A_1^*(x)$ be the transformed polynomial corresponding to the left subinterval of I^* and let $A_2^*(x)$ be the transformed polynomial corresponding to the right subinterval of I^* . Since I^* is an isolating interval for $A(x)$, one subinterval will contain a single real root of $A(x)$ and the other subinterval will not contain any real roots of $A(x)$. By Descartes' rule, the subinterval containing the root will have an odd number of coefficient sign variations and the other subinterval will have an even number of coefficient sign variations. Therefore, the subinterval corresponding to the bisection polynomial with an odd number of variations contains α and is selected. The algorithm **AFSIGND**, listed in Figure 44, is the same as **AFSIGN** except it uses this idea to select the subinterval containing α .

Theorem 75 (Computing Time of AFSIGND) *Assume $\deg(A(x)) = m$, $\deg(B(x)) = n$ with $m > n$ and $|A(x)|_1 = d$ and $|B(x)|_1 = e$. Let $I_1, \dots, I_r$ be isolating intervals for $\alpha_1, \dots, \alpha_r$ be the distinct real roots of $A(x)$. Then the time to compute $\text{sign}(B(\alpha_i))$ for $i = 1, \dots, r$ using **AFSIGN** is dominated by $m^5 L(de)^2$.*

Even though this algorithm has a better theoretical computing time than **AFSIGN** it does not perform as well in practice (see Table 25). The reason for this is that typically only a few bisections will be required and hence the size of the endpoints will be codominant with 1 or possibly $L(d)$ depending on the size of the initial

AFSIGND($A(x), I, B(x); I^*, s$)

[Algebraic field element sign, Descartes. Inputs: $A(x)$ is an integral polynomial. I is a standard isolating interval for a real root, α of $A(x)$. $B(x)$ is an integral polynomial such that $B(\alpha) \neq 0$. Outputs: I^* is a standard subinterval of I which does not contain any roots of $B(x)$. $s \in \{-1, 1\}$. $s = \text{sign}(B(\alpha))$.]

1. [α rational] if $\deg(A(x)) = 0 \vee W(I) = 0$ then { $I^* \leftarrow I$; $s \leftarrow \text{sign}(B(b))$.
2. [β rational.] if $\deg(B(x)) = 0$ then { $I^* \leftarrow I$; $s \leftarrow \text{sign}(\text{ldcf}(B(x)))$.
3. [Compute transformed polynomial $\tilde{A}(x)$ whose roots in $(0, 1)$ correspond to the roots of $A(x)$ in I .] $I = (a/2^h, (a+1)/2^h]$; $\tilde{A}(x) = T_{-a}H_{2^h}(A(x))$.
4. [Compute transformed polynomial $\tilde{B}(x)$ whose roots in $(0, 1)$ correspond to the roots of $B(x)$ in I .] $\tilde{B}(x) = T_{-a}H_{2^h}(B(x))$.
5. [Obtain an isolating interval for alpha containing no roots of $B(x)$ and evaluate the sign of $B(x)$ at the midpoint of I^* .] $I^* \leftarrow I$;
 repeat {
 FIRST2($I^*; a^*, b^*$); $c^* \leftarrow \text{RIB}(I^*)$; $B^*(x) \leftarrow T_{-1}R(\tilde{B}(x))$;
 if $\text{var}(B^*(x)) = 0 \wedge x \not\models B^*(x) = 0$ then { $s \leftarrow \text{sign}(B(w^*))$; return };
 $\tilde{A}(x) \leftarrow H_2(\tilde{A}(x))$; $\tilde{B}(x) \leftarrow H_2(\tilde{B}(x))$; $A_1^*(x) \leftarrow T_{-1}R(\tilde{A}(x))$;
 if $\text{even}(\text{var}(A_1^*(x))) \wedge x \not\models A_1^*(x)$ then {
 $a^* \leftarrow c^*$; $\tilde{B}(x) \leftarrow \tilde{B}(x+1)$ }
 else {
 $b^* \leftarrow c^*$; }
 $I^* \leftarrow (a^*, b^*)$; } ■

Figure 44: **AFSIGND** Algebraic Field Element Sign, Using Descarte's Rule

isolating interval. Therefore the cost of the polynomial evaluations will be dominated by $m^2L(d) + m^2$ or $m^2L(d)^2$ whereas the cost of the polynomial translation by one will be dominated by $m^3 + m^2L(d)$ or $m^3L(d)$.

Alternatively, interval arithmetic can be used to determine if $B(x)$ has any roots in the interval I . If the interval $B(I)$ does not contain zero then $B(x)$ does not have any roots in I and the sign of $B(\alpha)$ is equal to the sign of either endpoint of $B(I)$. Either Horner's method or the mean value theorem method can be used to evaluate $B(I)$ (see Section 5.1.4). The algorithm **AFSIGNI** uses Horner's method to compute $B(I)$ and the algorithm **AFSIGNIMVT** uses the mean value theorem approach to compute $B(I)$.

A bound on the computing time of either of the interval arithmetic algorithms can be obtained using Theorem 65 and a theorem due to Collins and Loos which gives a bound on how close $B(\alpha)$ can be to zero. As soon as the width of the interval $B(I^*)$ is less than this bound then we know that $0 \notin B(I^*)$ and the sign can be computed.

Theorem 76 (Collins and Loos) *Let $\deg(A(x)) = m$, $\deg(B(x)) = n$, $|A(x)|_1 = d$, and $|B(x)|_1 = e$. Assume $A(\alpha) = 0$ and $B(\alpha) \neq 0$. Then $|A(\alpha)| > 1/2(d+1)^{-n}e^{-m}$.*

PROOF. This bound is obtained from a lower root bound on $R(y) = \text{res}_x(A(x), y - B(x))$. By Theorem 14 $R(y) = a_m^n \prod_{i=1}^m y - B(\alpha_i)$. A lower root bound is obtained from a root bound for the inverted polynomial $y^m R(1/y)$ ■

Theorem 77 (Computing Time of AFSIGNI) *Assume $\deg(A(x)) = m$, $\deg(B(x)) = n$ with $m > n$ and $|A(x)|_1 = d$ and $|B(x)|_1 = e$. Let $A(\alpha) = 0$ and I be*

isolating intervals for α . Assume $B(\alpha) \neq 0$. Then the time to compute $\text{sign}(B(\alpha))$ using **AFSIGNI** is dominated by $m^5L(de)^3$.

PROOF. The algorithm terminates when zero is not contained in the interval $B(I^*)$. By Theorem 76 this is true when $W(B(I^*)) < 2(d+1)^ne^m$. Since, by Theorem 65, $W(B(I^*)) < md|I|^mW(I)$, this will be true when $W(I^*) < cd^ne^m$ for some constant c . Therefore the number of steps is dominated by $mL(de)$. Since the largest interval endpoints are dominated by $mL(de)$, each interval evaluation is dominated by $m^4L(de)^2$. Therefore the total time is dominated by $m^5L(de)^3$ ■

This bound also applies to **AFSIGNIMVT**. Even though both interval algorithms have the same maximum computing time bound, they behave differently in practice. The mean value theorem approach converges to the range faster than Horner's method. Nonetheless, Horner's method may produce the sign with fewer bisections. Furthermore, the cost of evaluating the range using the mean value theorem is slightly larger than using Horner's method. However, if many bisections are required, the mean value theorem approach will be better. This is because the mean value theorem approach converges faster and because if the monotonicity check succeeds the range can be computed by evaluating at the endpoints.

The mean value theorem approach can be modified so that it bisects the isolating interval for α until no roots of $B'(x)$ are contained in I and hence $B(x)$ is monotonic. If $B(x)$ is monotonic, we can check if $B(x)$ has any roots in I by computing the signs of $B(x)$ at the endpoints of I . If this idea is recursively applied to $B'(x)$ to check if $B'(x)$ has any roots in I , we obtain an algorithm due to Rump [44]. The algorithm

AFSIGNDS, listed in Figure 45, uses this idea to compute the sign of $B(\alpha)$.

AFSIGNDS($A(x), I, B(x); S, I^*$)

[Algebraic field element sign at derivative sequence. Inputs: $A(x)$ is an irreducible integral minimal polynomial for α . I is an isolating interval for α . $B(x)$ is an integral polynomial with $\gcd(A(x), B(x)) = 1$. Outputs: $S = (s_0, \dots, s_m)$ is a list of signs with $s_i = \text{sign}(B^i(\alpha))$.]

1. [B rational.] if $B(x) = 0$ then { $S \leftarrow (0)$; $I^* \leftarrow I$; return }; if $\deg(B(x)) = 0$ then { $S \leftarrow (\text{sign}(\text{ldcf}(B(x))))$; $I^* \leftarrow I$; return }.
2. [Recursion.] **AFSIGNDS**($A(x), I, B'(x); S', I^*$).
3. [refine I.] **FIRST2**($I^*; a^*, b^*$); $s_0 \leftarrow \text{sign}(A(a^*))$; $s_1 \leftarrow \text{sign}(A(b^*))$; $t_0 \leftarrow \text{sign}(B(a^*))$; $t_1 \leftarrow \text{sign}(B(b^*))$; while $t_0 * t_1 \leq 0$ do { $c^* \leftarrow \text{RIB}(a^*, b^*)$; $s \leftarrow \text{sign}(A(c^*))$; $t \leftarrow \text{sign}(B(c^*))$; if $s_1 = 0 \vee s \cdot s_1 < 0$ then { $a^* \leftarrow c^*$; $s_0 \leftarrow s$; $t_0 \leftarrow t$ } else { $b^* \leftarrow c^*$; $s_1 \leftarrow s$; $t_1 \leftarrow t$ } }; $I^* \leftarrow (a^*, b^*)$; $S \leftarrow \text{COMP}(t_1, S')$ ■

Figure 45: **AFSIGNDS** Algebraic Field Element Sign at Derivative Sequence

The algorithm **AFSIGNDS** assumes that $B^k(\alpha) \neq 0$ for all k . Even if $(A(x)$ and $B(x))$ are relatively prime, this may not be true for all of the derivatives; however, if it is assumed that $A(x)$ is irreducible and $\deg(B(x)) < \deg(A(x))$ then the desired property is true. If we do not want to make this assumption, we can use **AFZTEST** to determine if $B^k(\alpha) = 0$. If α is a root of $B^k(x)$ of even multiplicity, $B^{k-1}(x)$ will not be monotonic on the interval I^* no matter how many bisections are performed. In this case, the tangent construction in Section 4.3.1 can be used to refine I^* until $B^{(k-1)}(x)$ does not have any roots in I^* .

AFSIGNDS has essentially the same computing time as **AFSIGN** and the interval arithmetic based algorithms. However, in practice, **AFSIGNDS** can be more

costly, especially as the degree increases, since each of the derivatives must be evaluated at the endpoints of I^* . Nonetheless, **AFSIGNDS** has the benefit that the signs of all of the derivatives are computed as a side effect.

Theorem 78 (Computing Time of AFSIGNDS) *Assume $\deg(A(x)) = m$, $\deg(B(x)) = n$ with $m > n$ and $|A(x)|_1 = d$ and $|B(x)|_1 = e$. Let $A(\alpha) = 0$ and I be isolating intervals for α . Assume $B(\alpha) \neq 0$. Then the time to compute $\text{sign}(B(\alpha))$ using **AFSIGNI** is dominated by $m^8 + m^5 L(d)^3$.*

PROOF. [43] ■

Tables 25 and 26 compare the various sign algorithms discussed in this section for random inputs. Five random irreducible polynomials, $A(x)$, of degree m with k bit coefficients were generated. The real roots of $A(x)$ were isolated using the algorithm **IPRIS**. The number of real roots for the ten random $A(x)$ is denoted by C . For each real algebraic number field $Q(\alpha)$ defined by $A(x)$ and an isolating interval I for α , the signs of ten random elements of were computed. A random element of $Q(\alpha)$ is a random polynomial, $B(x)$, of degree less than the degree of $A(x)$. The average computing time, averaged over the $10C$ sign computations, for each algorithm is listed along with the average number of bisections required. These experiments show that very few bisections are required for random polynomials.

Tables 27 and 28 only include sign computations that required more than five bisections. For a given m and k , N is the number of sign computations, in our experiment, that required more than five bisections. In all of the experiments, the maximum number of bisections was nine.

Table 25: Empirical Computing Times (in ms) for Random Algebraic Sign Computation

m	k	C	AFSIGN	B_1	AFSIGND
10	10	12	5.0	0.53	7.2
10	155	10	16.4	1.07	24.0
10	300	10	23.8	0.80	33.8
20	10	20	16.7	0.55	26.8
20	155	10	50.2	1.15	80.3
20	300	14	68.2	0.69	110.7
30	10	16	39.4	0.77	66.0
30	155	12	102.0	1.20	169.4
30	300	16	133.8	0.63	222.7

Table 26: Empirical Computing Times (in ms) for Random Algebraic Sign Computation

m	k	C	AFSIGNI	B_3	AFSIGNIMVT	B_4	AFSIGNDS	B_5
10	10	12	8.3	0.65	8.1	0.64	13.5	1.69
10	155	10	19.3	1.36	23.6	1.40	48.7	1.90
10	300	10	26.1	1.07	31.7	1.05	84.5	1.74
20	10	20	23.4	0.73	24.1	0.75	73.7	1.81
20	155	10	50.8	1.50	60.8	1.58	217.7	2.75
20	300	14	50.8	0.92	67.6	1.01	386.3	2.96
30	10	16	43.9	1.08	44.9	1.08	220.5	2.97
30	155	12	83.5	1.60	105.9	1.77	513.9	2.89
30	300	16	82.6	0.88	110.0	0.98	837.6	2.44

Table 27: Empirical Computing Times (in ms) for Random Algebraic Sign Computation

m	k	N	AFSIGN	B_1	AFSIGND
10	155	5	40.7	1.07	57.0
10	300	3	53.3	0.80	71.1
20	10	1	105	9	175
20	155	5	120	6	206
20	300	4	188	6.25	300
30	10	8	122	5.25	207
30	155	10	225	5.8	361
30	300	3	301	5.3	464

Table 28: Empirical Computing Times (in ms) for Random Algebraic Sign Computation

m	k	N	AFSIGNI	B_3	AFSIGNIMVT	B_4	AFSIGNDS	B_5
10	155	5	52	1.36	51	1.40	58	1.90
10	300	3	73	1.07	66	1.05	965	1.74
20	10	1	170	0.73	53	0.75	75	1.81
20	155	5	176	1.50	114	1.58	244	2.75
20	300	4	226	0.92	205	1.01	383	2.96
30	10	8	163	1.08	106	1.08	185	2.97
30	155	10	264	1.60	211	1.77	517	2.89
30	300	3	316	0.88	302	0.98	904	2.44

5.3.1 Multiple Extensions

Any of the algorithms in this section can be used to compute the sign of an element in $Q(\alpha_1, \dots, \alpha_r)$ defined by a tower of extensions. The algorithm **ATFSIGN** computes the sign of $B(\alpha_1, \dots, \alpha_r)$ using Descartes' rule of signs, polynomial transformations and recursive sign computations in $Q(\alpha_1, \dots, \alpha_{r-1})$. The polynomial transformations T_{-a} , T_{-1} , and H_2 are performed using the algorithms **IPTR**, **IPTR1**, and **IPBHT** from Section 5.2.3. The polynomial evaluation $A(\alpha_1, \dots, \alpha_{r-1}, c^*)$ is performed using **IPBREI** from Section 5.2.3. Either the algorithm **AFSIGN** or **AFSIGND** can be used for the base case. The maximum computing time of **ATFSIGN** can be obtained in the same way the computing time bound was obtained for **AFSIGN**.

Theorem 79 (Computing Time of ATFSIGN) *Assume $\deg(A_1(x)) = m_1$, $\deg(A_2(\alpha, x_2)) = m_2$ and $|A_1(x)|_1 = d_1$, $|A_2(\alpha_1, x)|_1 = d_2$ and $|B(\alpha_1, \alpha_2)|_1 = e$. Then the time to compute $\text{sign}(B(\alpha_1, \alpha_2))$ for all real conjugates of α and β using **ATFSIGN** is dominated by $m_1^{13}m_2^8L(d)^4$. If **ATFSIGN** follows the approach of **AFSIGND** then the maximum computing time is reduced to $m_1^{11}m_2^6L(d)^3$.*

PROOF. The proof is similar to the proof of Theorem 74. In this case Davenport's theorem is applied to the norm of $A(\alpha_1, x_2)B(x_1, x_2)$ to obtain a bound on the number of bisections for a particular sign computation, and a bound on hk the number of bisections for all of the conjugates, k , times the level h . A bound for the norm gives a bound for the algebraic polynomial $B(\alpha_1, x_2)$. The bound obtained in this way is $m_1^2m_2L(de)$ (Theorem 82).

ATFSIGN($r, M, I, B(x_1, \dots, x_r); I^*, s$)

[Algebraic tower of fields element sign. Inputs: $M = (A_r(x_1, \dots, x_r), \dots, A_1(x_1))$ is a list of integral polynomials such that $A_i(\alpha_1, \dots, \alpha_i) = 0$, and $A_i(\alpha_1, \dots, \alpha_{i-1}, x_i)$ is irreducible. $I = (I_r, \dots, I_1)$ is a list of standard intervals, such that I_i is an isolating interval for a real root α_i of $A_i(x_i)$. $B(x_1, \dots, x_r)$ is an integral polynomial. Outputs: $I^* = (I_r^*, \dots, I_1^*)$ is a list of refined isolating intervals such that I_r^* does not contain any roots of $B(\alpha_1, \dots, \alpha_{r-1}, x_r)$. $s = \text{sign}(B(\alpha_1, \dots, \alpha_r))$.]

1. [$B(x_1, \dots, x_r) = 0$] if $B(x_1, \dots, x_r) = 0$ then { $s \leftarrow 0$; $I^* \leftarrow I$; return }.
2. [Base case.] $r' \leftarrow r - 1$; **ADV**($M; A, M'$); **ADV**($I; I_0, I'$); if $r' = 0$ then { **AFSIGN**($A, I_0, B; s, I_0^*$); $I^* \leftarrow (I_0^*)$; return }.
3. [$B(\alpha_1, \dots, \alpha_{r-1}, x_r) \in Q(\alpha_1, \dots, \alpha_{r-1})$.] if $\deg(B(\alpha_1, \dots, \alpha_{r-1}, x_r)) = 0$ then { $I^* \leftarrow I$; $s \leftarrow \text{sign}(\text{ldcf}(B(\alpha_1, \dots, \alpha_{r-1}, x_r)))$ }.
4. [Compute transformed polynomial $\tilde{B}(x)$ whose roots in $(0, 1)$ correspond to the roots of $B(x)$ in I .] $I_0 = (a/2^h, (a+1)/2^h]$; $\tilde{B}(\alpha_1, \dots, \alpha_{r-1}, x_r) \leftarrow T_{-a}H_{2^h}(B(\alpha_1, \dots, \alpha_{r-1}, x_r))$.
5. [Obtain an isolating interval for alpha containing no roots of B and evaluate the sign of B at its bisection point.] $t \leftarrow \text{sign}(B(\alpha_1, \dots, \alpha_{r-1}, b))$; $I_0^* \leftarrow I_0$;
 repeat {
 FIRST2($I_0^*; a^*, b^*$); $c^* \leftarrow \text{RIB}(I_0^*)$;
 $B^*(\alpha_1, \dots, \alpha_{r-1}, x_r) \leftarrow T_{-1}R(\tilde{B}(\alpha_1, \dots, \alpha_{r-1}, x_r))$;
 if $\text{var}(B^*(\alpha_1, \dots, \alpha_{r-1}, x_r)) = 0 \wedge x_r \nparallel B^*(\alpha_1, \dots, \alpha_{r-1}, x_r)$ then {
 $s \leftarrow \text{sign}(B(\alpha_1, \dots, \alpha_{r-1}, w^*))$; return };
 $\tilde{B}(\alpha_1, \dots, \alpha_{r-1}, x_r) \leftarrow H_2(\tilde{B}(\alpha_1, \dots, \alpha_{r-1}, x_r))$; $w \leftarrow \text{sign}(A(\alpha_1, \dots, \alpha_{r-1}, c^*))$;
 if $tw < 0 \vee t = 0$ then {
 $a^* \leftarrow c^*$; $\tilde{B}(\alpha_1, \dots, \alpha_{r-1}, x_r) \leftarrow T_{-1}(\tilde{B}(\alpha_1, \dots, \alpha_{r-1}, x_r))$ }
 else {
 $b^* \leftarrow c^*$; $t \leftarrow w$ };
 $I_0^* \leftarrow (a^*, b^*)$; } ■

Figure 46: **ATFSIGN** Algebraic Tower of Fields Sign

By Theorem 73, the time for the k polynomial translations at any level is dominated by $m_1^3 m_2^4 L(de)$. Since the number of levels is dominated by $m_1^2 m_2 L(de)$, the total time for translations is dominated by $m_1^5 m_2^5 L(de)^2$.

The time to calculate $\text{var}(B^*(\alpha_1, x_2))$ is dominated by the cost of computing the m_2 coefficient signs. By Theorem 74 this is dominated by $km_2(m_1^5(m_2 h)^3)$, which is dominated by $m_1^{11} m_2^7 L(de)^3$. Therefore the total time for computing $\text{var}(B^*(\alpha_1, x_2))$ is dominated by $m_1^{13} m_2^8 L(de)^4$. If instead the approach of **AFSIGND** is taken the total cost of computing $\text{var}(B^*(\alpha_1, x_2))$ is dominated by $m_1^{11} m_2^6 L(de)^3$.

In either approach the time for the coefficient sign variation computations dominates the cost of the evaluation of $A(\alpha_1, \dots, \alpha_{r-1}, c^*)$ ■

5.4 Algebraic Polynomial Norm

The norm is used to derive and analyze algorithms for isolating the real roots of a polynomial with real algebraic number coefficients.

Let

$$A(x) = a_m \prod_{i=1}^m (x - \alpha_i)$$

be a defining polynomial for $\alpha = \alpha_1$. Let $B(\alpha, y)$ be a polynomial in $Q(\alpha)[y]$, with $\deg(B(\alpha, y)) = n$. The *Norm* of $B(\alpha, y)$ is equal to

$$B^*(y) = \text{res}_\alpha(A(\alpha), B(\alpha, y)),$$

which by Theorem 14 is equal to

$$a_m^{m-1} \prod_{i=1}^m B(\alpha_i, y).$$

If $B(\alpha, \beta) = 0$ then $B^*(\beta) = 0$. Therefore, a root bound for $B^*(y)$ is a root bound for $B(\alpha, y)$, and $\text{sep}(B(\alpha, y)) \leq \text{sep}(B^*(y))$. Using this idea we can obtain a root bound and a bound on the minimum root separation of an algebraic polynomial. First we need a bound on the size of the coefficients of $B^*(y)$.

Theorem 80 (Norm Coefficient Bound) *Assume $A(\alpha) = 0$, where $A(x)$ is an integral polynomial of degree m . Let $B(\alpha, y)$ be a polynomial in $Z[\alpha][y]$ of degree n . Further assume that $|A(x)|_1 \leq d$ and $|B(\alpha, y)|_1 \leq e$. If $B^*(y)$ is equal to the norm of $B(\alpha, y)$, then $\deg(B^*(y)) = mn$ and $|B^*(y)|_1 \leq |A(x)|_1^{m-1} |B(\alpha, y)|_1^m$.*

PROOF. This is a special case of Theorem 16 ■ This theorem can be used to obtain a dominance relation for a root bound and the minimum root separation of $B(\alpha, y)$.

Theorem 81 (Algebraic Polynomial Root Bound) *Assume $A(\alpha) = 0$, where $A(x)$ is an integral polynomial of degree m . Let $B(\alpha, y)$ be a polynomial in $Z[\alpha][y]$ of degree n . Further assume that $|A(x)|_1 \leq d$ and $|B(\alpha, y)|_1 \leq e$. Then the root bound of $B(\alpha, y)$ is dominated by $(de)^m$.*

PROOF. Apply Cauchy's root bound formula (Theorem 8) and Theorem 80 to the norm of $B(\alpha, y)$ ■

Theorem 82 (Algebraic Polynomial Root Separation) *Assume $A(\alpha) = 0$, where $A(x)$ is an integral polynomial of degree m . Let $B(\alpha, y)$ be a polynomial in $Z[\alpha][y]$ of degree n . Further assume that $|A(x)|_1 \leq d$ and $|B(\alpha, y)|_1 \leq e$. Then $\log(\text{sep}(B(\alpha, y))^{-1})$ is dominated by $m^2 n L(de)$. Also let $\alpha_1, \dots, \alpha_r$ be the real roots of $A(x)$. Then the product of the distances between the real roots of $\prod_{i=1}^r B(\alpha_i, y)$ is dominated by $(de)^{m^2 n}$.*

The same bound is obtained for the product of the distances between the pairs of complex conjugate roots.

PROOF. Apply The integral polynomial root separation theorem (Theorem 2) to the norm of $B(\alpha, y)$ ■

The norm can be computed using Collins multivariate polynomial resultant algorithm.

Theorem 83 (Computing Time Norm($B(\alpha, y)$)) *Assume $A(\alpha) = 0$, where $A(x)$ is an integral polynomial of degree m . Let $B(\alpha, y)$ be a polynomial in $Z[\alpha][y]$ of degree n . Further assume that $|A(x)|_1 \leq d$ and $|B(\alpha, y)|_1 \leq e$. The norm of $B(\alpha, y)$ can be computed in time dominated by $m^4nL(de) + m^3n^2L(de) + m^3nL(de)^2$.*

PROOF. [10] ■

5.5 Algebraic Polynomial Root Bound Computation

In this section we investigate several algorithms for computing a root bound of a polynomial whose coefficients are in a real algebraic number field $Q(\alpha)$. The algorithms for algebraic polynomial root bound computation are based on the algorithms for integral polynomials in Section 3.4. The algorithm **IUPPRBD**, which is based on repeated homothetic transformations and Descartes' rule of signs can be carried over to polynomials with real algebraic number coefficients. The algorithm **IUPRB** based on Knuth's root bound formula (Theorem 9) can be applied to the norm of $B(\alpha, y)$. To apply Knuth's formula directly to the polynomial $B(\alpha, y)$, we need to

obtain rational bounds for the coefficients of $B(\alpha, y)$. In section 5.5.1 we present two algorithms for obtaining an interval, with arbitrarily small width, containing an element of $Q(\alpha)$.

The first algorithm, **AFPNRB** (Algebraic Field Polynomial Norm Root Bound), computes a root bound for $B(\alpha, y)$ by computing a root bound for the norm of $B(\alpha, y)$. The root bound of the norm can be computed with any root bound algorithm for integral polynomials. We use the algorithm **IUPRB**. Since a root bound for the norm is a root bound for $B(\alpha_i, y)$ for any real conjugate, α' , of α , this root bound applies to all of the conjugate polynomials $B(\alpha', y)$.

Theorem 84 (Computing Time of Norm Based Root Bound) *Assume $A(\alpha) = 0$ and $\deg(A(x)) = m$, and assume $|A(x)|_1 \leq d$. Also assume $\deg(B(\alpha, y)) = n$ and $|B(\alpha, y)|_1 \leq e$. Then the computing time of **AFPNRB**($A(x), I, B(\alpha, y)$) is dominated by $m^4nL(de) + m^3n^2L(de) + m^3nL(de)^2$.*

PROOF. By theorem 83 the time to compute $\text{Norm}(B(\alpha, y))$ is dominated by $m^4nL(de) + m^3n^2L(de) + m^3nL(de)^2$. By Theorem 80, $|\text{Norm}(B(\alpha, y))|_1 \preceq mL(de)$, hence by Theorem 39 the time to compute a root bound of $\text{Norm}(B(\alpha, y))$ is dominated by $m^2nL(de)$, which is dominated by the cost of computing the norm ■

The second algorithm, **AFPPRBD** (Algebraic Field Polynomial Positive Root Bound using Descartes' Rule), is the same as the integral polynomial algorithm **IP-PRBD** except the algorithms in Section 5.2.3 must be used for the polynomial transformations and a sign algorithm in Section 5.3 must be used to count the number of coefficient sign variations.

Theorem 85 (Computing Time of Descartes' Rule Based Root Bound)

Assume $A(\alpha) = 0$, $\alpha \in I$, and $\deg(A(x)) = m$, and assume $|A(x)|_1 \leq d$. Also assume $\deg(B(\alpha, y)) = n$ and $|B(\alpha, y)|_1 \leq e$. Then the computing time of AFPRBD is dominated by $m^8 n^3 L(de)^3$

PROOF. The proof is similar to the proof of the integral polynomial case (Theorem 40). Since by Theorem 81 a root bound is dominated by $mL(de)$, the number of homothetic transformations is dominated by $mL(de)$. After h homothetic transformations, the size of the coefficients of the transformed polynomial are dominated by $nh + L(e)$. Therefore, after h homothetic transformations, the time for the translation by one is dominated by $m(n^3 h + n^2 L(e))$, which is dominated by $m^2 n^3 L(de)$ since h is dominated by $mL(de)$. Therefore, since there at most $mL(de)$ iterations, the total time for all of the translations is dominated by $m^3 n^3 L(de)^2$. Using Theorem 75, the time to compute the n coefficient sign variations at the h -th iteration is dominated by $n(m^5 (nh)^2)$, which is dominated by $m^7 n^3 L(de)^2$. Therefore the total time for sign computations is dominated by $m^8 n^3 L(de)^3$ which dominates the cost for the entire algorithm ■

The third algorithm begins by computing an interval polynomial (i.e. a polynomial with interval coefficients) that contains the algebraic polynomial $B(\alpha, y)$. This means that each interval coefficient contains the corresponding coefficient of $B(\alpha, y)$. The leading coefficient of the interval polynomial is refined until it does not contain zero. This makes the degree of the interval polynomial well defined and is necessary to compute a root bound for the interval polynomial. Rational bounds are then computed for each of the interval coefficients to obtain a rational polynomial $\overline{B}(x) = \sum_{j=0}^n \overline{b}_j y^j$

is computed such that $|\bar{b}_j| \geq b_j(\alpha)$ for $0 \leq j < n$ and $|\bar{b}_n| < |b_n(\alpha)|$. A root bound for the rational polynomial is then computed using the algorithm **IUPRB**.

The algorithm **AFPIRB** (Algebraic Field Polynomial Interval Root Bound) uses this approach to compute a root bound for an algebraic polynomial. **AFPIRB** uses the algorithm **AFINTMVT** in Section 5.5.1 to compute the interval coefficients. The algorithm **AFPIRB1** uses the algorithm **INFAF** to compute the interval coefficients. The interval coefficients are refined until their widths are less than one.

Theorem 86 (Computing Time of Interval Based Root Bound Algorithm)

*Assume $A(\alpha) = 0$, $\alpha \in I$, and $\deg(A(x)) = m$, and assume $|A(x)|_1 \leq d$. Also assume $\deg(B(\alpha, y)) = n$ and $|B(\alpha, y)|_1 \leq e$. Then the computing time of **AFPIRB** is dominated by $m^5 n L(de)^3$*

PROOF. The computing time for **AFPIRB** is dominated by the cost of computing the interval polynomial, which by Theorem 89 is dominated by $nm^5 L(de)^3$ (the time for computing an interval for each of the n coefficients) ■

This algorithm can be improved by retaining the refined isolating interval for α after each call to **AFINTMVT**. Thus refinements to the isolating interval are not redone. The algorithm **AFPIRBI** incorporates this improvement. Not only does this lead to a significant practical improvement, it also leads to an improved computing time bound.

Theorem 87 (Computing Time of Improved Interval Based Root Bound)

Assume $A(\alpha) = 0$, $\alpha \in I$, and $\deg(A(x)) = m$, and assume $|A(x)|_1 \leq d$. Also assume

$\deg(B(\alpha, y)) = n$ and $|B(\alpha, y)|_1 \leq e$. Then the computing time of **AFPIRBI** is dominated by $m^5L(de)^3 + m^4nL(de)^2$

PROOF. The same bound on the number of bisections required to refine the isolating interval I for a single call to **AFINTMVT** applies to the number of refinements required by all of the coefficients. Therefore the total number of interval evaluations in **AFINTMVT** is dominated by $mL(de) + n$, the number required to make I small enough plus the number to check each of the n coefficients. Since each interval evaluation is dominated by $m^4L(de)^2$, the total time is dominated by $m^5L(de)^3 + m^4nL(de)^2$ ■

Table 29 compares the five root bound algorithms discussed in this section. The comparisons are for random polynomials. Five irreducible random polynomials of degree m were generated whose real roots define a real algebraic number field $Q(\alpha)$. For each random real algebraic number field five random polynomials with coefficients in $Q(\alpha)$ were generated. The different root bound algorithms were then applied to these polynomials. B is the average number of bisections required, by the preceding algorithm, to refine the isolating interval for α . All times are in milliseconds.

Table 29: Empirical Comparison of Algebraic Polynomial Root Bound Algorithms

m	n	AFPNRB	AFPRBD	AFPIRB1	AFPIRB	B	AFPIRBI	B
5	5	1816 ms	330 ms	8383 ms	3304 ms	187	1270 ms	34
5	10	4347 ms	867 ms	16396 ms	6597 ms	335	1850ms	34
10	5	12903 ms	463 ms	119540 ms	6516 ms	166	2342 ms	31

5.5.1 Bounds for Real Algebraic Numbers

Given an element, $\beta = B(\alpha)$, of a real algebraic number field $Q(\alpha)$ we wish to find an interval that contains $B(\alpha)$ and whose width is less than ϵ . The first approach uses algorithm the **ANFAF** to convert the representation of β as an element of $Q(\alpha)$ to the representation of a real algebraic number. This involves computing a defining polynomial and isolating interval for β . Once this change of representation has been performed, the isolating interval can be refined so that its width is less than ϵ . The algorithm **INFAF**, listed in Figure 47, uses this approach.

$$J^* \leftarrow \text{INFAF}(A, I, B, k; M, J)$$

[Interval from Algebraic field element. Inputs: $A(x)$ is an integral defining polynomial for a real algebraic number α . I is a standard isolating interval for α . $\beta = B(\alpha)$ is an element of $Z[\alpha]$. k is an integer. Outputs: $M(x)$ is an integral defining polynomial for $B(\alpha)$. J is a standard isolating interval for β . $W(J) < 2^k$.]

1. [Compute defining polynomial and isolating interval.]
 $\text{ANFAF}(A(x), I, B(x); M(x), J)$.
2. [Refine isolating interval.] $J \leftarrow \text{Refine}(M(x), J, k)$ ■

Figure 47: **INFAF** Interval from Algebraic Field Element

Theorem 88 (Computing Time of INFAF) *Let $\deg(A(x)) = m > \deg(b(x))$, and $|A(x)|_1 \leq d$ and $|b(x)|_1 \leq e$. Then the computing time of **INFAF** is dominated by $m^7 L(de)^2$*

PROOF. The computing time is dominated by the time to isolate the real roots of the defining polynomial, $M(y) = \text{gsfd}(\text{Norm}(y - B(\alpha)))$, for β . By Theorem 80,

the size of the coefficients of the defining polynomial for β are dominated by $mL(de)$. Therefore, by Theorem 53, the time to isolate the real roots of the $M(y)$ is dominated by $m^7L(de)^2$ ■

The second approach is to use interval arithmetic to compute the desired interval containing $B(\alpha)$. The algorithm **AFINTMVT**, listed in Figure 48, uses the mean value theorem method of obtaining an interval containing the range of $B(I)$, which contains $B(\alpha)$. In the mean value theorem evaluation of the range of $B(I)$, $B'(I)$ is evaluated using Horner's method. If $0 \notin B'(I)$ then we know that $B(x)$ is monotonic on I and the range can be computed from the endpoints. This usually applies if the interval is refined a sufficient number of times.

Theorem 89 (Computing Time of AFINTMVT) *Let $\deg(A(x)) = m > \deg(b(x))$, and $|A(x)|_1 \leq d$ and $|b(x)|_1 \leq e$. Then the computing time of **AFINTMVT** is dominated by $m^5L(de)^3$*

PROOF. The proof is similar to the proof of Theorem 77 ■

The computing time of **INFAP** is typically dominated by the cost of computing the defining polynomial, and this cost is typically more than the entire algorithm **AFINTMVT**. However, **INFAP** typically produces intervals with smaller sized endpoints.

5.6 Algebraic Polynomial GCD Computation

In this section we review some computing time bounds for computing the greatest common divisor of two polynomials with coefficients in an algebraic number field.

AFINTMVT($A, I, B, k; J, I^*$)

[Algebraic field element interval using mean value theorem interval evaluation. Inputs: $A(x)$ is an integral defining polynomial for a real algebraic number α . I is a standard isolating interval for α . $B(\alpha)$ is an element of $Z[\alpha]$. k is an integer. Outputs: J is a standard interval containing $B(\alpha)$. $W(J) < 2^k$. I^* is the refined isolating interval for α .]

1. [Compute range.]
 - repeat {
 - FIRST2**(I^* ; a, b);
 - if monotonic then {
 - $c \leftarrow B(a)$; $d \leftarrow B(b)$;
 - if $c < d$ then $J \leftarrow (c, d)$ else $J \leftarrow (d, c)$ }
 - else {
 - $m \leftarrow \mathbf{RIB}(I^*)$; $J \leftarrow B(m) + B'(I^*)(I^* - m)$; }
 - until $W(J) < 2^k$ ■

Figure 48: **AFINTMVT** Algebraic Field Element Interval, Using Mean Value Theorem Interval Evaluation

Many of these results are from Langemyr's thesis [30], which should be consulted for a detailed discussion along with complete proofs. These theorems will be needed in the analysis of the root isolation algorithms in Chapter VI. Besides using these algorithms to compute greatest common divisors, the subresultant PRS algorithm will be used to compute a Sturm sequence.

The subresultant PRS of two polynomials in $Z[\alpha][y]$ can be computed by first computing the corresponding bivariate integral PRS and then reducing the coefficients by the minimal polynomial for α . The PRS in $Z[x, y]$ can be computed using a modular algorithm. An algorithm using this idea was presented by Rubald in his thesis [42]. Using our observation in Section 4.1.1 we can use a modular algorithm to compute the subresultant PRS which does not encounter any unlucky primes. This observations leads to a simpler algorithm with a better computing time bound than Rubald's algorithm.

In addition to providing computing time bounds, this section also presents an algorithm with important practical implications for the derivative sequence based root isolation algorithms. We present an algorithm which may be able to quickly determine that two polynomials are relatively prime. This is important for the derivative sequence algorithms since they compute many gcds of polynomials which are almost always relatively prime.

5.6.1 Algebraic Polynomial GCDs and the Subresultant PRS Algorithm

By the fundamental theorem of PRSs, the gcd of two polynomials whose coefficients are in $Z[\alpha, y]$ is similar to the non-vanishing subresultant of lowest degree. Therefore, an associate of the gcd can be computed using the subresultant PRS. There are three approaches to computing the subresultant PRS over $Z[\alpha, y]$. The first approach is to compute directly with arithmetic in $Z[\alpha]$. In this approach elements of $Z[\alpha]$ are always reduced modulo the defining polynomial for α . This can be done provided α is an algebraic integer, since in this case the defining polynomial, $A(x)$, of α is monic and reductions modulo $A(x)$ preserve the integer coefficients. The second approach is to compute the subresultant PRS over $Z[x, y]$ and reduce the coefficients modulo $A(x)$ after the PRS has been completed. The correctness of this approach is based on Theorem 45 which asserts that the image under a homomorphism (in this case evaluation at α) of the subresultant is the same as the subresultant of the images. The third approach is the same as this approach except that the subresultant PRS over $Z[x, y]$ is computed with a modular algorithm.

We begin with a theorem which bounds the coefficients of the subresultant.

Theorem 90 (Subresultant Coefficient Bound) *Assume $A(x)$ is a monic integral polynomial such that $A(\alpha) = 0$. Further assume $\deg(A(x)) = m$ and $|A(x)|_1 \leq d$. Also assume $B_1(\alpha, y)$ and $B_2(\alpha, y)$ are polynomials with coefficients in $Z[\alpha]$ with $\deg_y(B_i(\alpha, y)) = n$ and $|B_i(\alpha, y)|_1 \leq e$. Then $|S_k(B_1(\alpha, y), B_2(\alpha, y))|_1 \preceq d^{mn}e^n$.*

PROOF. By Theorem 16, $|S_k(B_1(x, y), B_2(x, y))|_1 \preceq e^n$, and after reducing the coeffi-

cients by taking pseudo-remainders, Theorem 42 implies that $|S_k(A(\alpha, y), B(\alpha, y))|_1 \preceq e^n d^{mn}$ ■

Theorem 91 (Time for Algebraic Subresultant PRS) *Assume $A(x)$ is a monic integral polynomial such that $A(\alpha) = 0$. Further assume $\deg(A(x)) = m$ and $|A(x)|_1 < d$. Also assume $B_1(\alpha, y)$ and $B_2(\alpha, y)$ are polynomials in $Z[\alpha][y]$ with $\deg(B_i(\alpha, y)) = n$ and $|B_i(\alpha, y)|_1 < e$. The time to compute the subresultant PRS of $B_1(\alpha, y)$ and $B_2(\alpha, y)$, using arithmetic in $Z[\alpha]$ is dominated by $m^5 n^5 L(de)^2$.*

PROOF. [30] ■

Theorem 92 (Time for Integral Subresultant PRS and Reduction) *Assume $A(x)$ is a monic integral polynomial such that $A(\alpha) = 0$. Further assume $\deg(A(x)) = m$ and $|A(x)|_1 < d$. Also assume $B_1(\alpha, y)$ and $B_2(\alpha, y)$ are polynomials in $Z[\alpha][y]$ with $\deg(B_i(\alpha, y)) = n$ and $|B_i(\alpha, y)|_1 < e$. The time to compute the subresultant PRS of $B_1(\alpha, y)$ and $B_2(\alpha, y)$, by computing the integral subresultant PRS of $B_1(x, y)$ and $B_2(x, y)$ and reducing the coefficients modulo $A(x)$, is dominated by $n^8 m^2 L(de)^2 + n^3 m^3 L(de)^2$.*

PROOF. [30] ■

Instead of computing the bivariate integral subresultant PRS using integer arithmetic, a modular algorithm can be used (see [42]). We use a modification of Rubald's algorithm similar to the algorithm IPNSPRSM in Section 4.1. This algorithm computes the subresultant PRS of the first kind and does not have to deal with unlucky

primes. The bivariate subresultants are computed by using the evaluation homomorphism and the result is obtained by interpolation. See [10], for a description of this approach to computing resultants.

Theorem 93 (Time for Modular Bivariate Subresultant) *Assume $B_1(x, y)$ and $B_2(x, y)$ are polynomials with coefficients in $Z/(p)$ where p is a prime assumed to be codominant with one. Let $\deg_x(B_i(x, y)) = m$ and $\deg_y(B_i(x, y)) = n$. Then the time to compute $S_k(B_1(x, y), B_2(x, y))$ is dominated by $m(n - k)n^2 + m^2(n - k)^2(k + 1)$.*

PROOF. [10] ■

Theorem 94 (Time for Integral Bivariate Subresultant PRS) *Assume $B_1(x, y)$ and $B_2(x, y)$ are polynomials with integral coefficients Let $\deg_x(B_i(x, y)) = m$ and $\deg_y(B_i(x, y)) = n$, and $|B_i(x, y)|_1 \leq d$. Then the time to compute the integral subresultant PRS of $B_1(x, y)$ and $B_2(x, y)$ using a modular algorithm is dominated by $m^2n^4L(d) + mn^4L(d)^2$.*

PROOF. The computing time of the algorithm is dominated by the cost of computing the modular subresultants, which can be obtained from [10] ■

After the integral Bivariate Subresultant PRS has been computed the coefficients need to be reduced to obtain the PRS over $Z[\alpha, y]$.

Theorem 95 (Time for Integral Subresultant PRS and Reduction) *Assume $A(x)$ is a monic integral polynomial such that $A(\alpha) = 0$. Further assume $\deg(A(x)) = m$ and $|A(x)|_1 < d$. Also assume $B_1(\alpha, y)$ and $B_2(\alpha, y)$ are polynomials in $Z[\alpha][y]$*

with $\deg(B_i(\alpha, y)) = n$ and $|B_i(\alpha, y)|_1 < e$. The time to compute the subresultant PRS of $B_1(\alpha, y)$ and $B_2(\alpha, y)$ using a modular algorithm followed by reduction of the coefficients modulo $A(x)$ is dominated by $m^2n^4L(e) + mn^4L(e)^2 + m^3n^5L(d)^2 + m^2n^2L(d)L(e)$.

PROOF. Use Theorem 94 to bound the cost of computing the subresultant PRS over $Z[x, y]$ and then Theorem 43 to bound the cost of performing the reductions ■

If only the gcd is required the modular algorithm due to Lagemyr and McCallum ([32]) can be used. This algorithm uses modular images in $Z_p[\alpha, y]$.

5.6.2 Fast Test for Relatively Prime Polynomials

Frequently two polynomials are relatively prime. The algorithm **AFPRP**, listed in Figure 49, uses a modular resultant computation to attempt to determine if two polynomials are relatively prime. Let $B_1(\alpha, y)$ and $B_2(\alpha, y)$ be two polynomials in $Z[\alpha, y]$. Since the smallest nonvanishing subresultant is similar to the gcd, if the resultant does not vanish then the polynomials are relatively prime. Furthermore, if $\text{res}(B_1(\alpha, y), B_2(\alpha, y)) \not\equiv 0 \pmod{p}$ then $\text{res}(B_1(\alpha, y), B_2(\alpha, y)) \neq 0$.

Theorem 96 (Time for AFPRP) Assume $A(x)$ is the integral minimal polynomial for α and that $\deg(A(x)) = m$ and $|A(x)|_\infty \leq d$. Further assume that $B_1(\alpha, y)$ and $B_2(\alpha, y)$ are two polynomials in $Z[\alpha, y]$ and $\deg(B_i(\alpha, y)) \leq n$ and $|B_i(\alpha, y)|_\infty \leq e$. Then the computing time for **AFPRP** is dominated by $mn^3 + m^2n^2 + mL(d) + mnL(e)$.

$$t \leftarrow \mathbf{AFPRP}(M(x), A(\alpha, y), B(\alpha, y))$$

[Algebraic field polynomial relative primality test. Inputs: $M(x)$ is the integral minimal polynomial for an algebraic number α . $A(\alpha, y)$ and $B(\alpha, y)$ are elements of $Z[\alpha][y]$. Outputs: If it can be shown that $\gcd(A(\alpha, y), B(\alpha, y)) = 1$, then $t = 1$, else $t = 0$.]

1. [Get small prime.] $P \leftarrow \text{List of Primes}$; $m \leftarrow \deg(A(\alpha, y))$; $n \leftarrow \deg(B(\alpha, y))$;
 repeat {
 $\mathbf{ADV}(P; p, P)$; $A^*(\alpha, y) \leftarrow \phi_p(A(\alpha, y))$; $B^*(\alpha, y) \leftarrow \phi_p(B(\alpha, y))$;
 $M^*(x) \leftarrow \phi_p(M(x))$ }
 until $\deg(A^*(\alpha, y)) = m \wedge \deg(B^*(\alpha, y)) = n$.
2. [Compute mod p resultant.] $R^*(\alpha) \leftarrow \text{res}(A^*(\alpha, y), B^*(\alpha, y))$.
3. [Check if $R(x) \equiv 0 \pmod{A^*(x)}$.] $\mathbf{MPQR}(1, p, R^*(x), M^*(x); Q^*(x), R^*(x))$;
 if $R^*(x) \neq 0$ then $t \leftarrow 1$ else $t \leftarrow 0$ ■

Figure 49: **AFPRP** Algebraic Field Polynomial Relative Primality Test

PROOF. The time to apply the modular homomorphism to $A(x)$ is dominated by $mL(d)$ and the time to apply the modular homomorphism to $B_i(x, y)$ is dominated by $mnL(e)$. The proof is completed by using Theorem 93 to bound the time for computing the modular resultant ■

CHAPTER VI

Real Root Isolation of Polynomials with Real Algebraic Number Coefficients

In this chapter we discuss several algorithms for isolating the real roots of a polynomial with real algebraic number coefficients. The first section discusses a norm based algorithm that relies on integral polynomial real root isolation. The next three sections discuss the Sturm sequence, derivative sequence, and coefficient sign variation algorithms. Section 6.5 discusses an important improvement which can be used to dramatically reduce the maximum computing time bounds of the algorithms. This improvement also has practical significance. The last section discusses a version of the coefficient sign variation method which uses interval arithmetic. While this is not an algorithm, since it may not terminate, it can be used to isolate the roots of many polynomials. This approach has the potential to significantly speed up the coefficient sign variation method for polynomials with real algebraic number coefficients.

6.1 Reduction to Real Root Isolation of Integral Polynomials

In this section we present an algorithm for isolating the real roots of a polynomial with real algebraic number coefficients. This algorithm uses the norm to reduce the

problem to real root isolation of a polynomial with integral coefficients. Let $A(x)$ be an integral polynomial of degree m and let I be an isolating interval for a real root, α , of $A(x)$. Let $Q(\alpha)$ be the real algebraic number field defined by $A(x)$ and I . Let $B(\alpha, y)$ be a polynomial in $Q(\alpha)[y]$. By clearing the denominators we can assume that $B(\alpha, y)$ has coefficients in $Z[\alpha]$. We further assume that $B(\alpha, y)$ is squarefree.

Let $B^*(y) = \text{Norm}(B(\alpha, y))$. By definition, the set of real roots of $B^*(y)$ contains the real roots of $B(\alpha, y)$. Therefore, any isolating interval for $B^*(y)$ either contains no roots of $B(\alpha, y)$ or is an isolating interval for $B(\alpha, y)$. If $I = (a, b]$ is a strong isolating interval for the norm, $B^*(y)$, then I is an isolating interval for $B(\alpha, y)$ if and only if $B(\alpha, a) \cdot B(\alpha, b) < 0$ or $B(\alpha, b) = 0$. The assumption that the intervals be strong isolating intervals prevents the possibility that $B(\alpha, a) = 0$. The algorithm **AFPRIN**, listed in Figure 50, is based on these ideas.

In step 1 the greatest squarefree divisor of $B^*(y)$ is computed. This is necessary since many integral polynomial real root isolation algorithms require that their inputs be squarefree and since the norm of a squarefree polynomial may not be squarefree. As a trivial example, the norm of a polynomial with integer coefficients, $B(y)$, is a power of $B(y)$. **AFPRIN** can use any real root isolation algorithm for isolating the real roots of the norm. Our results in Section 4.5 suggest that the best algorithm for isolating the real roots of an integral polynomial is the algorithm based on Descartes' rule of signs, **IPRICS**.

It is possible to reduce the number of sign computations in step 3 of **AFPRIN**. If $I_j = (a_j, b_j]$ is an isolating interval for $B^*(y)$, then $(b_{j-1}, b_j]$ is also an isolating interval

$$L \leftarrow \text{AFPRIN}(A(x), I, B(\alpha, y))$$

[Algebraic Field Polynomial Root Isolation using the Norm. $A(x) \in \mathbb{Z}[x]$ is the integral minimal polynomial for the real algebraic number α . I is a standard isolating interval for α . $B(\alpha, y) \in \mathbb{Z}[\alpha][y]$. $B(\alpha, y)$ is squarefree. $L = (I_1, \dots, I_r)$ is a list of isolating intervals for the real roots of $B(\alpha, y)$.]

1. [Compute Norm.]
 $B^*(y) \leftarrow \text{gsfd}(\text{Norm}(B(\alpha, y)))$.
2. [Isolate roots of norm.]
 $L^* \leftarrow \text{RootIsolation}(B^*(y)); L^* \leftarrow \text{StronglyDisjoint}(L^*)$.
3. [Find Isolating Intervals in $\mathbb{Q}(\alpha)$.]
 for $i = 1, \dots, \text{LENGTH}(L^*)$ do
 if $\text{sign}(B(\alpha, a_i))\text{sign}(B(\alpha, b_i)) < 0 \vee B(\alpha, b_i) = 0$ then $L \leftarrow \text{append}(L, I_i)$ ■

Figure 50: **AFPRIN** Algebraic Field Polynomial Root Isolation Using the Norm

for $B^*(y)$. Therefore, it is necessary only to compute the signs of $B(\alpha, y)$ at the right endpoints of the isolating intervals of $B^*(y)$. This should reduce the computing time of step 3 by a factor of two. However, our timing results in Section 6.1.2 show that most of the computing time of **AFPRIN** is taken in steps 1 and 2

Let $\alpha_1 = \alpha, \dots, \alpha_r$ be the real roots of $A(x)$. If it is necessary to isolate the real roots of $B(\alpha_i, y)$ for all of the real conjugates of α , then steps 1 and 2 need only to be performed once. Only the signs computed in step 3 will vary for the different conjugates of α . Furthermore, it may be desirable to compute the norm for other purposes such as Trager's norm based factorization algorithm [47] and in the extension phase of Collins CAD algorithm [18].

We conclude this section with a couple of examples. Let $A(x) = -243x^3 - 788x^2 -$

$81x + 127$. $A(x)$ has three real roots $\alpha_1 \in (-4, -3)$, $\alpha_2 \in (-1, -1/2)$, and $\alpha_3 \in (0, 8)$.

Let

$$\begin{aligned} B(\alpha, y) = & (-939\alpha^2 + 368\alpha + 132)y^5 + (-998\alpha^2 + 708\alpha - 390)y^4 + \\ & (-911\alpha^2 - 312\alpha - 816)y^3 + (-744\alpha^2 + 746\alpha + 128)y^2 + \\ & (70\alpha^2 + 228\alpha + 690)y + (154\alpha^2 - 497\alpha - 981) \end{aligned}$$

Then the norm of $B(\alpha, y)$,

$$\begin{aligned} B^*(y) = & 25185725213793y^{15} + 72713164384084y^{14} - 177394713044081y^{13} - \\ & 892097513273738y^{12} - 1312937664383661y^{11} - 914197381707058y^{10} - \\ & 455628485956737y^9 - 797573289930686y^8 - 1044631532627044y^7 - \\ & 490250309364596y^6 - 887833555060015y^5 - 311032888270900y^4 + \\ & 478257788099863y^3 - 385062110586642y^2 - 113237479085714y + \\ & 93050266221881, \end{aligned}$$

has degree 15 and maximum coefficient size 51 bits. $B^*(y)$ is squarefree and has five real roots: $\beta_1 \in J_1 = (-4, -2)$, $\beta_2 \in J_2 = (-2, -1)$, $\beta_3 \in J_3 = (-1, 0)$, $\beta_4 \in J_4 = (0, 2)$, and $\beta_5 \in J_5 = (2, 4)$. The computation of the norm required 83 ms, determining that $B^*(y)$ is squarefree required 17 ms, and isolating the real roots of $B^*(y)$ required 50 ms using the coefficient sign variation algorithm **IPRICS**. The maximum of the heights of the search trees used by **IPRICS** to isolate the positive and negative real roots of $B^*(y)$ is 4, the total number of interior nodes is 5, and the total number of leaf nodes is 7. The signs of $\text{sign}(B(\alpha_1, y))$ evaluated at the

endpoints of the intervals $J_i = (a_i, b_i)$ are

$$((+, +), (+, -), (-, +), (+, -), (-, -)).$$

The signs of $\text{sign}(B(\alpha_2, y))$ evaluated at the endpoints of these intervals are

$$((+, -), (-, -), (-, -), (-, -), (-, -)).$$

The signs of $\text{sign}(B(\alpha_3, y))$ evaluated at the endpoints of these intervals are

$$((-, -), (-, -), (-, -), (-, -), (-, +)).$$

Therefore, J_2 , J_3 , and J_4 are isolating intervals for the three real roots of $B(\alpha_1, y)$, J_1 is an isolating interval for the real root of $B(\alpha_2, y)$, and J_5 is an isolating interval for the real root of $B(\alpha_3, y)$. The average time (averaged over the three conjugates of α) to evaluate and compute the signs of $B(\alpha_i, y)$ at the endpoints of the isolating intervals of $B^*(y)$ was 39 ms.

As a second example, consider the polynomial

$$\begin{aligned} B(\alpha, y) = & (-399\alpha^2 - 112\alpha + 55)x^5 + (-881\alpha^2 - 141\alpha - 381)x^4 + \\ & (-172\alpha^2 + 958\alpha + 363)x^3 + (-889\alpha^2 + 726\alpha - 443)x^2 + \\ & (-89\alpha^2 - 573\alpha - 132)x + (-188\alpha^2 - 549\alpha + 725), \end{aligned}$$

where the defining polynomial for α is the same as in the first example. The root isolation algorithm **AFPRIN** has vastly different behavior on this example. In this case the norm has nine real roots, the maximum height and number of nodes of the search tree for **IPRICS** required to isolate the real roots of the norm are 22 and 64

respectively. The time to compute the norm was 83 ms and the time to isolate the real roots of the norm was 333 ms. Unlike the previous example, it is more costly to isolate the real roots of the norm than to compute the norm.

6.1.1 Computing Time

Theorem 97 (Comp. Time of AFPRIN) *Let $m = \deg(A(x))$, $n = \deg(B(\alpha, y))$, $|A(x)|_1 \leq d$, and $|B(\alpha, y)|_1 \leq e$. Then the computing time of AFPRIN is dominated by $m^{10}n^5 \log(de)^2$.*

PROOF. By Theorem 83 step (1) takes time dominated by $m^4nL(de) + m^3n^2L(de) + m^3nL(de)^2$. By Theorem 80, the size of the coefficients of the norm are dominated by $mL(de)$. Even though the coefficients of the greatest squarefree divisor may be larger than those of the norm (Theorem 4), the root separation stays the same. Hence by Theorem 53 the time to isolate the real roots of the norm is dominated by $m^7n^5L(de)^2$.

There are at most mn polynomial evaluations in step (3), which by Theorem 71, take time dominated by $m^6n^5L(de)^2$. Since, after evaluation, the size of the coefficients of $B(\alpha, b_i)$ are dominated by $nH \preceq m^2n^2L(de)$, (H is the bound on the height of the search tree obtained from Mahler's root separation theorem), Theorem 75 implies that the sign computations in step (3) take time dominated by $mn(m^5(m^2n^2L(de))^2) = m^{10}n^5L(de)^2$ ■

6.1.2 Empirical Behavior

This section reports some empirical results for the algorithm AFPRIN. These results also give some more empirical data for the integral polynomial real root isolation

algorithm **IPRICS**. For these experiments the algorithm **IPRICS** is used on polynomials obtained from the norm of random polynomials in $Q(\alpha)[y]$, where α is a root of a random integral polynomial $A(x)$.

For each degree m five random irreducible polynomials, $A(x)$, defining a real algebraic number field $Q(\alpha)$ were generated. Each polynomial $A(x)$ was randomly generated with its coefficients uniformly distributed between -2^{10} and 2^{10} . The real roots of $A(x)$ were isolated and the number of real roots gives the number of conjugate real algebraic number fields $Q(\alpha)$ that were used. For each real algebraic number field $Q(\alpha)$, ten random polynomials $B(\alpha, y)$ of degree n and coefficient size 10 bits were generated. The algorithm **AFPRIN** was then used to isolate the real roots of $B(\alpha, y)$. C is the total number of conjugate fields $Q(\alpha)$. In other words, the number of real roots of the five random polynomials, $A(x)$, is equal to C . Therefore the total number of real algebraic polynomial root isolations is equal to $10C$. r is the average number of real roots of the $10C$ polynomials $B(\alpha, y)$. The column **Total** refers to the average time (in ms) for the algorithm **AFPRIN**. The column **Norm** is the average time required to compute the norm, and the column **RI** is the average time to isolate the roots of the norm with the algorithm **IPRICS**.

R is the average number of real roots of the norm. H is the average height and N is the average number of nodes of the search trees used by **IPRICS** to isolate the roots of the norm. Recall that the height is defined to be the maximum of the heights of the trees used to isolate the negative and positive roots, and the number of nodes is the sum of the number of nodes in the trees used to isolate the negative

and positive roots.

Table 30: Empirical Behavior of AFPRIN for Random Polynomials

m	C	n	r	R	H	N	Norm	RI	Total
5	7	5	1.80	2.52	5.9	16.6	438	188	740
5	7	10	2.17	3.04	5.8	17.5	1504	607	2323
5	9	15	2.44	4.40	7.0	21.8	2771	1800	4854
5	5	20	3.04	3.04	6.7	19.9	4682	3041	7912
10	10	5	1.94	3.88	7.2	22.4	4178	1089	5427
15	11	5	1.71	3.76	7.4	23.6	15908	3227	19577
20	16	5	1.81	5.80	8.6	29.6	41556	8744	51000

6.2 An Algorithm Based on Sturm Sequences

In this section we discuss the use of the Sturm sequence based algorithm for real root isolation of polynomials with real algebraic number coefficients. We can restrict our attention to polynomials with coefficients in $Z[\alpha]$. Since $Z[\alpha]$ is not a unique factorization domain, we can not use the primitive PRS as we did in the integer case. Therefore, we use the monic PRS and the subresultant PRS. In both cases the size of the coefficients of the Sturm sequence are even more daunting than for integral polynomial Sturm sequences.

Since the other algorithms for root isolation are far superior to the Sturm sequence algorithm, we will not go to great length to tune this algorithm. We will only implement the monic PRS algorithm and make a few comments about various subresultant approaches. Our main use of the subresultant algorithms is in proving maximum computing times. In practice the subresultant algorithms should be some-

what better than the monic PRS algorithm; however, the improvement should not be enough to make any of the algorithms comparable with the other methods of real root isolation.

Definition 33 (Negative Monic PRS) *The negative monic PRS*

$\{B_1(x), B_2(x), \dots, B_t(x), B_{t+1}(x) = 0\}$ is defined by $e_i B_i(x) = Q_i(x) B_{i+1}(x) + f_i B_{i+2}(x)$, where $e_i = 1$, $f_i = -|\text{ldcf}(\text{rem}(B_i(x), B_{i+1}(x)))|$, and $B_{i+2}(x)$ is the monic associate of $\text{rem}(B_i(x), B_{i+1}(x))$.

The negative monic PRS of $A(\alpha, x)$ and $A'(\alpha, x)$ is a Sturm sequence for $A(\alpha, x)$. The negative monic PRS is computed using arithmetic in $Q(\alpha)$ and elements of $Q(\alpha)$ are represented by $r\bar{b}(\alpha)$ where r is a rational number and $\bar{b}(\alpha)$ is an integral polynomial in α whose degree is less than the degree of the minimal polynomial of α .

Let $A(x)$ and $B(\alpha, x)$ be the polynomials defined in the first example in Section 6.1. Recall that $A(x)$ defines three real algebraic number fields $Q(\alpha_i)$ corresponding to the three real roots of $A(x)$. Let $\alpha_1 \in (-4, -3)$, $\alpha_2 \in (-1, -1/2)$, and $\alpha_3 \in (0, 8)$ be the real roots of $A(x)$. The negative monic PRS for $B(\alpha_1, y)$ is (in the listing $a = \alpha$)

$$\begin{aligned} B_1(a, y) = & y^5 + \\ & ((-2/8395241737931) (8703424344141 a^2 + 27530834303077 a - \\ & 2854197106428)) y^4 + \\ & ((-1/66104265653) (245975430996 a^2 + 823972704404 a + \\ & 148116618987)) y^3 + \end{aligned}$$

$$\begin{aligned}
& ((2/8395241737931) (431745164739 a^2 + 2472797861120 a + \\
& 7429594287721)) y^2 + \\
& ((2/8395241737931) (11554397407575 a^2 + 38141198275271 a + \\
& 7607569973298)) y \\
& ((-1/8395241737931) (31695420025152 a^2 + 105033388175063 a + \\
& 24581236476494))
\end{aligned}$$

$$\begin{aligned}
B_2(a,y) = & y^4 + \\
& (-8/41976208689655) (8703424344141 a^2 + 27530834303077 a - \\
& 2854197106428)) y^3 + \\
& ((-3/330521328265) (245975430996 a^2 + 823972704404 a + \\
& 148116618987)) y^2 + \\
& ((4/41976208689655) (431745164739 a^2 + 2472797861120 a + \\
& 7429594287721)) y + \\
& ((2/41976208689655) (11554397407575 a^2 + 38141198275271 a + \\
& 7607569973298))
\end{aligned}$$

$$\begin{aligned}
B_3(a,y) = & -y^3 + \\
& ((-3/57989198296428723515704940641) (4532099453971837817648224140 a^2 \\
& - 10050592364509658569629470929 a + 4534056577251920533163804299)) y^2 + \\
& ((4/57989198296428723515704940641) (29572447281618591542332382499 a^2 +
\end{aligned}$$

$$81163596289492851697423497596 a - 11075352868883965873681710169)) y + \\ ((-1/115978396592857447031409881282) (278615691033823904102281452492 a^2 \\ + 902373266990588024397539628503 a - 80050930350169195951183655870))$$

$$B_4(a,y) = -y^2 + \\ ((-3/4604354787846296977659990393206087351588808119397250676482) \\ (55142589758837102075688537906915478031955070365434071011 a^2 + \\ 802501718053210619668118478241945078136330025864211682552 a + \\ 1746253466658815314875499269022769276241765323276805415577)) y + \\ ((-1/657764969692328139665712913315155335941258302771035810926) \\ (634949747959051957053667573170636937929672517780333544529 a^2 + \\ 1498274896795198881747923330844442000074172544114975098228 a - \\ 1671772063048856211636547030900107059675121364171295169276))$$

$$B_5(a,y) = -y + \\ ((-1/17688900693658518873394624355605356275358399614046023891744 \\ 490510840413082824733729300409) \\ (766407615231947914528757119090119267259589977106202806302143004 \\ 6275219177656082620846868 a^2 + \\ 1899141245264869572791191902677421587938512341331313202537651504 \\ 0596352137569828222143089 a - \\ 5747497604347745863777607626311066879319140482035310092999804740$$

881791723489744734292069a))

$$B_6(a, y) = -1$$

Since the signs of the leading coefficients of the Sturm sequence are

$$(+, +, -, -, -, -),$$

$B(\alpha, x)$ has $V_\infty - V_{-\infty} = 4 - 1 = 3$ real roots. Once a Sturm sequence has been computed an analog of algorithm **IPRIST** can be used to isolate the real roots. This algorithm called **AFPRIST** (Algebraic field polynomial root isolation using Sturm sequences), required a search tree of height 3 with 2 interior nodes and 3 leaf nodes.

The Sturm sequences over the real conjugates fields of $Q(\alpha)$ defined by the intervals $(-1, -1/2)$ and $(0, 8)$ are both equal to

$$(B_1(x), B_2(x), -B_3(x), -B_4(x), B_5(x), -B_6(x)).$$

Therefore, it is easy to verify that $B(\alpha_2, x)$ and $B(\alpha_3, x)$ each have one real root.

There are several disadvantages to using the monic PRS. The first is that rational arithmetic is required. The second is that we haven't been able to derive a bound for the size of the coefficients in the Sturm sequence. The third is that there does not seem to be a way of converting the monic PRS to the corresponding negative monic PRS after the sequence has been computed. This has the disadvantage that the Sturm sequence has to be recomputed for each conjugate field. All three of these difficulties can be eliminated by using the subresultant PRS. However, the subresultant PRS still has large coefficients and requires considerable computation time.

The algorithm **AFPRIST** was used to isolate the real roots of the 70 random polynomials, $B(\alpha, y)$, from Section 6.1 with $m = 5$ and $n = 5$. The average computing time was approximately 18 seconds. Of these 18 seconds, 17 seconds were spent computing the monic Sturm sequence, 335ms were spent computing a root bound, and 700ms were spent isolating the real roots.

6.2.1 Subresultant Sturm Sequences

In this section we sketch several algorithms for computing the subresultant Sturm sequence for polynomials with real algebraic number coefficients. In this section we will assume that the input polynomial actually has coefficients in $Z[\alpha]$ and that α is an algebraic integer. This will enable all subsequent arithmetic to be performed in $Z[\alpha]$, thereby avoiding costly rational arithmetic.

We begin with an algorithm that converts the subresultant PRS to a negative subresultant PRS. Using this algorithm it is possible to convert the subresultant PRS to a Sturm sequence for all of the conjugates of α without having to recompute the PRS.

Let $G_1, G_2, \dots, G_{r+1}$ be the subresultant PRS. By the definition of the subresultant PRS, $e_i = g_i^{\delta_i+1}$, $f_1 = 1$ and $f_i = -g_i(-h_i)^{\delta_i}$ for $i > 1$, where $h_1 = 1$ and $h_i = g_i^{\delta_{i-1}} h_{i-1}^{\delta_{i-1}+1}$ for $i > 1$. Let $s_1 = 1$, $s_2 = 1$, and $s_{i+2} = \text{sign}(e_i)\text{sign}(f_i)\text{sign}(s_i)$ for $i > 2$. Then by Theorem 9, $s_i G_i$ is a negative subresultant PRS. Using the definition of the subresultant PRS it is possible to compute s_i from the signs of g_i . As a special case, if the subresultant PRS is normal, it is especially easy to compute s_i .

Theorem 98 *Let $G_1, G_2, \dots, G_{r+1}$ be a normal subresultant PRS. Then $s_i G_i$ is a negative PRS when $s_i = 1$ for $i \equiv 3, 4 \pmod{4}$, and $s_i = -1$ for $i \equiv 1, 2 \pmod{4}$.*

PROOF. Since the PRS is normal, $\delta_i = 1$ for all i . Therefore, from the definition of the subresultant PRS, e_i and f_i are both positive. Thus $s_{i+2} = -s_i$ and the result follows from induction on i using since $s_1 = 1$ and $s_2 = 1$ ■

If the PRS is not normal, the cost of converting to a negative PRS is dominated by the cost of computing the signs of the leading coefficients of the subresultant PRS.

Theorem 99 (Time for Conversion to Negative Subresultant PRS)

Assume $A(x)$ is a monic integral polynomial such that $A(\alpha) = 0$. Further assume $\deg(A(x)) = m$ and $|A(x)|_1 < d$. Also assume $B(x, y)$ is an integral polynomial with $\deg(B(x, y)) = n$ and $|B(x, y)|_1 < e$. The time to convert the subresultant PRS for $B(\alpha, y)$ over $Q(\alpha)$ to a negative subresultant PRS is dominated by $m^7 n^3 L(de)^2$.

PROOF. By Theorem 80 the lengths of the leading coefficients in the subresultant PRS are dominated by $mnL(de)$. Therefore, by Theorem 75 the time to compute the signs of the leading coefficients is dominated by $nm^5(mnL(de))^2 \preceq m^7 n^3 L(de)^2$ ■

As discussed in Section 5.6.1, there are several ways of computing the subresultant PRS over $Q(\alpha)$. We will use a modular algorithm to obtain a theoretical computing time bound for computing a Sturm sequence. Using Theorem 95 as a bound to compute the Subresultant PRS and Theorem 99 to bound the time to convert the Subresultant PRS to a negative PRS, we obtain the following bound on the time to compute the Subresultant Sturm sequence.

Theorem 100 (Time for Subresultant Sturm Sequence) *Assume $A(x)$ is a monic integral polynomial such that $A(\alpha) = 0$. Further assume $\deg(A(x)) = m$ and $|A(x)|_1 < d$. Also assume $B(x, y)$ is an integral polynomial with $\deg(B(x, y)) = n$ and $|B(x, y)|_1 < e$. The time to compute the subresultant Sturm sequence for $B(\alpha, y)$ over $Q(\alpha)$ is dominated by $m^7 n^3 L(de)^2 + m^2 n^4 L(e) + mn^4 L(e)^2 + m^3 n^5 L(d)^2$*

If the subresultant PRS is normal we can use Theorem 98 to convert to a negative PRS and the cost of the Sturm sequence construction will be dominated by the cost of computing the subresultant PRS.

Using the SAC-2 implementation of the subresultant PRS for integral polynomials, we computed the subresultant PRS over $Z[x, y]$ for the polynomials used in the previous section to time the algorithm **AFPRIST**. The average computing time to compute the integral subresultant PRS was approximately 3 seconds and the time to reduce the coefficients of the subresultants modulo the defining polynomial for α was approximately 2.5 seconds.

6.3 An Algorithm Based on Rolle's theorem and the Derivative Sequence

In this section we discuss the use of the derivative sequence method for real root isolation applied to polynomials with real algebraic number coefficients. This approach was taken by S. Rump in his thesis [43]. Rump implemented the algorithm as described by Collins and Loos in [14]. This algorithm is the one currently used in SAC-2 Computer Algebra system. We briefly discuss this algorithm and some of its shortcomings. We also discuss a new implementation following the approach to the

derivative sequence method presented in Section 4.3.

Rump was able to obtain the following computing time bound for the Collins-Loos algorithm.

Theorem 101 (Computing Time Bound for the Collins-Loos Algorithm)

Let $m = \deg(A(x))$, $n = \deg(B(\alpha, y))$, and $|A(x)|_1, |B(\alpha, y)|_1 \leq d$. Then the computing time of the Collins-Loos Algorithm is dominated by $m^{13}n^{13} + m^{13}n^9L(d)^4$.

PROOF. [43] ■

We will be able to significantly improve this bound using our version of the Collins-Loos algorithm, our improved algorithm for computing the sign of a real algebraic number, and Davenport's theorem (Theorem 3).

The major drawback to the current implementation is that at each stage of the inductive process it requires the computation of the greatest squarefree divisor of the gcd of $A(x)$ and its derivative. If a PRS based algorithm is used to compute these gcds, the Collins-Loos algorithm can be significantly more time consuming than the Sturm sequence method, since a Sturm sequence can be computed in the time required to compute the first gcd in the Collins-Loos algorithm. To alleviate this problem we use the fast relative primality test discussed in Section 5.6. Since the gcd of $A(x)$ and $A'(x)$ is almost always one, this test usually eliminates the costly gcd computation.

Instead of using the tangent construction of the Collins-Loos algorithm, The signs of $A(x)$ at the roots of $A'(x)$ can be computed using any algebraic sign computation algorithm. As in the algorithm **IPRIDS** of Section 4.3, we use the sign algorithm based on Descartes' rule of signs. However, this requires the computation of sign

of an algebraic number in a multiple extension. For example, suppose that $A(\alpha, y)$ is a polynomial whose coefficients are in the real algebraic number field $Q(\alpha)$. If α' be a root of the derivative $A'(\alpha, y)$, the algorithm requires the computation of $\text{sign}(A(\alpha, \alpha'))$, an element of the multiple extension $Q(\alpha, \alpha')$. Note that we do not require that $A'(\alpha, y)$ be irreducible over $Q(\alpha)[y]$. All that is required is that $A(\alpha, \alpha') \neq 0$, and this is determined by checking whether $\text{gsfd}(\gcd(A(\alpha, y), A'(\alpha, y)))$ has a root in the isolating interval for α' . In particular, if $\deg(\gcd(A(\alpha, y), A'(\alpha, y))) = 0$, then we know that $A(\alpha, \alpha') \neq 0$. The algorithm **AFPRIDS**, listed in Figure 51, is the analog to the algorithm **IPRIDS**. Since we assume that $A(\alpha, y)$ is in $Z[\alpha][y]$, we can use the integral evaluation algorithm in Section 5.2.2. A further advantage of assuming that $A(\alpha, y)$ is in $Z[\alpha, y]$ is that we can use the integral primitive derivative sequence instead of the derivative sequence. That is, the integral primitive part is taken of each derivative. **AFPRIDS** also uses the fast relative primality test **AFPRP** to first check if $A(x)$ and $A'(x)$ are relatively prime before computing their gcd.

6.3.1 Computing Time

A bound on the maximum computing time of the algorithm **AFPRIDS** can be obtained in the same way as the bound for **IPRIDS** was obtained in Theorem 54. In this case we need to use the computing time bound of the multiple extension version of the algebraic sign algorithm (Theorem 79).

Theorem 102 (Comp. Time of AFPRIDS) *Let $m = \deg(A(x))$, $n = \deg(B(\alpha, y))$, and $|A(x)|_1, |B(\alpha, y)|_1 \leq d$. Then the computing time of **AFPRIDS** is dominated by $m^{11}n^{10} + m^{11}n^7L(d)^3$.*

$$L \leftarrow \mathbf{AFPRIDS}(M(x), I, A(\alpha, y))$$

[Algebraic field polynomial real root isolation using derivative sequence. Inputs: $M(x)$ is the integral minimal polynomial for a real algebraic number α . I is a standard isolating interval for α . $A(\alpha, y)$ is an element of $Z[\alpha][x]$ with $\deg(A(\alpha, y)) = m$. Outputs: $L = (I_1, m_1, \dots, I_r, m_r)$ of isolating intervals and multiplicities for the real roots of $A(\alpha, y)$. $I_j = (a_j, b_j]$ is a left open right closed binary rational interval containing a unique root, β_j , of $A(\alpha, y)$. m_j is the multiplicity of β_j . The intervals are strongly disjoint.]

1. [Initialize.] $\bar{A} \leftarrow \text{pp}(A(\alpha, y)); \mathbf{AFUPRBH}(M(x), I, A(\alpha, y); B_1, \bar{A});$
 $B_0 \leftarrow -B_1; L \leftarrow ()$.
2. [Recursion.]
 if $\deg(\bar{A}) = 1$ then { $L \leftarrow ((B_0, B_1), 1)$; return }
 else { $L' \leftarrow \mathbf{AFPRIDS}(M(x), I, A'(\alpha, y))$ }.
3. [Construct isolating intervals for A from isolating intervals for A'.]
 $b_0 \leftarrow B_0; \mathbf{AFSIGN}(M(x), I, \text{ldcf}(\bar{A}); s_0, I^*);$
 if $\text{odd}(\deg(\bar{A}))$ then $s_0 \leftarrow -s_0$;
 if $L' \neq ()$ then { $\mathbf{AFUPGC}(M(x), \bar{A}, A'; C, \bar{A}_1, \bar{A}'); \bar{C} \leftarrow \text{gsfd}(C)$ }
 else goto 7.
4. [$A(\alpha_i) = 0$] $\mathbf{ADV2}(L'; J', m', L'); \mathbf{FIRST2}(J'; a^*, b^*);$
 if $\deg(\bar{C}) > 0$ then { $\mathbf{AFSIGN}(M(x), I, \mathbf{IPBREI}(2, \bar{C}, 2, a^*); \bar{s}, I^*);$
 $\mathbf{AFSIGN}(M(x), I, \mathbf{IPBREI}(2, \bar{C}, 2, b^*); \bar{t}, I^*);$
 if $\bar{s} * \bar{t} < 0 \vee \bar{t} = 0$ then { $L \leftarrow \mathbf{COMP2}(m' + 1, J', L); s \leftarrow 0$; goto 6 } }.
5. [Compute sign of $A(\alpha_i)$ and apply Rolle's theorem.]
 $J' \leftarrow \mathbf{AFPSIFI}(M(x), I, A'(\alpha, y), J'); \mathbf{ATFSIGN}(2, (A', M), (J', I), \bar{A}; s, J^*);$
 $J' \leftarrow \mathbf{FIRST}(J^*); \mathbf{FIRST2}(J'; a^*, b^*);$
 if $s_0 s < 0$ then { $L \leftarrow \text{append}(L, (b_0, a^*), 1)$ }.
6. [Update.] $b_0 \leftarrow b^*; s_0 \leftarrow s$; if $L' \neq ()$ then goto 4.
7. [Check (α, ∞) . Finish.] $\mathbf{AFSIGN}(M(x), I, \text{ldcf}(\bar{A}); s, I^*);$
 if $s_0 s < 0$ then { $L \leftarrow \text{append}(L, (b_0, B_1), 1)$ }; $L \leftarrow \text{inverse}(L)$ ■

Figure 51: **AFPRIDS** Algebraic Field Polynomial Root Isolation Using the Derivative Sequence

PROOF. Since $L(|A^{(k)}(\alpha, y)|_1) \preceq n + L(d)$, Theorem 79 implies that all of the signs for $A^{(k)}(x)$ can be computed in time dominated by $m^{11}n^6(n + L(d))^3$. Since there are n derivatives, the theorem is proved ■

6.3.2 Empirical Behavior

We conclude this section by reporting the computing times of **AFPRIDS** on the random polynomials introduced in Section 6.1. For each degree m five random irreducible polynomials, $A(x)$, defining a real algebraic number field $Q(\alpha)$ were generated. Each polynomial $A(x)$ was randomly generated with its coefficients uniformly distributed between -2^{10} and 2^{10} . The real roots of $A(x)$ were isolated and the number of real roots gives the number of conjugate real algebraic number fields $Q(\alpha)$ that were used. For each real algebraic number field $Q(\alpha)$, ten random polynomials $B(\alpha, y)$ of degree n and coefficient size 10 bits were generated. The algorithm **AFPRIDS** was then used to isolate the real roots of $B(\alpha, y)$. C is the total number of conjugate fields $Q(\alpha)$. In other words, the number of real roots of the five random polynomials, $A(x)$, is equal to C . Therefore the total number of real algebraic polynomial root isolations is equal to $10C$. r is the average number of real roots of the $10C$ polynomials $B(\alpha, y)$, and the number in the column under the algorithm name, **AFPRIDS**, is the average time (in ms) to isolate the real roots of the $10C$ polynomials $B(\alpha, y)$.

The computing times in Table 31 can be compared to the times for the SAC-2 implementation of the Collins-Loos algorithm (**AFPRCL**) and a version of **AFPRIDS** that does not use the fast relative primality check **AFPRP**. Since both of these algorithms are significantly slower, we only timed them for the polynomials with $m = 5$

Table 31: Empirical Behavior of **AFPRIDS** for Random Polynomials

m	C	n	r	AFPRIDS
5	7	5	1.80	1154
5	7	10	2.17	6706
5	9	15	2.44	19909
5	5	20	3.04	47872
10	10	5	1.94	3589
15	11	5	1.71	6011
20	16	5	1.81	10802

and $n = 5$. For these polynomials, the average computing time of **AFPRCL** was approximately 43 seconds and the average computing time of **AFPRIDS** without the fast relative primality test was approximately 30 seconds. Both of these times, as anticipated, are slower than the corresponding times for the monic Sturm sequence algorithm **AFPRIST**.

6.4 An Algorithm Based on Descartes' Rule and Polynomial Transformations

In this section we present a version of the root isolation algorithm, based on Descartes' rule of signs, which applies to polynomials with real algebraic number coefficients. The algorithm is a direct analog of the algorithm for integral polynomials (**IPRICS**). The main difference is that the sign computations required to count the number of coefficient sign variations is significantly more costly. In fact, at least theoretically, the time for the sign computations dominates the cost of the entire algorithm. In practice, the cost of the polynomial translations is comparable to the cost of the sign

computations. Which subalgorithm accounts for more time depends upon the particular inputs. The last difference between this algorithm and its integral counterpart is the root bound computation. We chose to use the root bound computation which is also based on Descartes' rule of signs. Besides being faster, or at least comparable, to the other algebraic polynomial root bound algorithms, it has the added benefit of producing the initial transformed polynomial as a side effect.

The analog to algorithm **IPRICS** is **AFPRICS**, and the analog to the normalized algorithm **IPRINCS**, which isolates the roots of a polynomial in the interval $(0, 1)$, is **AFPRINCS**. The algorithms **AFPRICS** and **AFPRINCS** are listed in Figures 52 and 53.

It is interesting to compare this algorithm to the norm based algorithm, which uses **IPRICS** to isolate the roots of the norm. Let $A(x)$ and $B(\alpha, y)$ be the polynomials in the first example in Section 6.1. We traced the call to **AFPRICS** $(A(x), I, B(\alpha, y))$ for the three real roots of $A(x)$. For $\alpha_1 \in (-4, -3)$, $B(\alpha_1, y)$ has three real roots and the associated search tree has height 2, 4 nodes, and 3 leaf nodes. Furthermore, the total number of bisections required for all of the algebraic sign computations was 4. For $\alpha_2 \in (-1, -1/2)$, $B(\alpha_2, y)$ only has one real root and this was isolated after one bisection. The total number of bisections requires for sign computations was 13. For the final conjugate, $\alpha_3 \in (0, 8)$, $B(\alpha_3, y)$ also had only one real root. However, the height of the search tree was 3, the number of nodes was 8, and the number of leaf nodes was 5. Furthermore, 145 bisections were required for sign computations. Therefore, for all three conjugates a total of 15 nodes occurred, the maximum height

$$L \leftarrow \mathbf{AFPRICS}(M(x), I, A(\alpha, y))$$

[Algebraic field polynomial real root isolation, coefficient sign variation method. Inputs: $M(x)$ is the defining polynomial for a real algebraic number α . I is an isolating interval for α . $A(\alpha, y) \neq 0$ is a squarefree polynomial in $Z[\alpha, y]$. Outputs: $L = (I_1, \dots, I_r)$ is a list of isolating intervals for the real roots of $A(\alpha, y)$. $I_j = (a_j, b_j)$ is standard open or one-point binary rational interval and $a_1 \leq b_1 \leq \dots \leq a_r \leq b_r$.]

1. [Initialize and check if $A(0) = 0$.]
 if $x|A(x)$ then $\{ L_0 \leftarrow ((0, 0)); A(x) = x/A(x); \}$
 else $L_0 \leftarrow ()$;
 $L_p \leftarrow ()$; $L_n \leftarrow ()$.
2. [Isolate positive roots.]
 $\mathbf{AFUPRBH}(M(x), I, A(\alpha, y); b_p, \tilde{A}_p(\alpha, y))$;
 $L_p \leftarrow \mathbf{AFPRINCS}(M(x), I, \tilde{A}_p(\alpha, y), 0, 1)$;
 Scale intervals in L_p by b_p .
3. [Isolate negative roots.] $A_n(\alpha, y) \leftarrow \mathbf{IPNT}(2, \tilde{A}_p(\alpha, y), 2)$; $b_n \leftarrow -b_p$;
 $L_n \leftarrow \mathbf{AFPRINCS}(M(x), I, \tilde{A}_n(\alpha, y), 0, 1)$;
 Scale intervals in L_n by b_n .
4. [Combine.] $L \leftarrow \text{concat}(L_n, L_0, L_p)$ ■

Figure 52: **AFPRICS** Algebraic Field Polynomial Root Isolation, Coefficient Sign Variation Method

$$L \leftarrow \text{AFPRINCS}(M(x), I, A(\alpha, y), J)$$

[Algebraic field polynomial root isolation, normalized coefficient sign variation method. Inputs: Inputs: $M(x)$ is the defining polynomial for a real algebraic number α . I is an isolating interval for α . $A(\alpha, y) \neq 0$ is a squarefree polynomial in $Z[\alpha, y]$. $J = (a, b)$ is a standard interval. Outputs: $L = (I_1, \dots, I_r)$ is a list of isolating intervals for $T(A(x))$ in the interval $(0, 1)$, where T is the linear fractional transformation that maps (a, b) onto $(0, 1)$. $I_j = (a_j, b_j)$ is either a standard open or one-point interval and $a_1 \leq b_1 \leq \dots \leq a_r \leq b_r$.]

1. [Initialize and check if $A(a) = 0$.]
 $a \leftarrow \text{LeftEndpoint}(I)$; $b \leftarrow \text{RightEndpoint}(I)$;
 if $x|A(x)$ then $\{ L_0 \leftarrow ((a, a)); A(x) \leftarrow A(x)/x \}$
 else $L_0 \leftarrow ()$.
2. [Base case.] $A^*(\alpha, y) \leftarrow \text{IPTR1}(2, \text{PRT}(A(\alpha, y)), 2)$;
 $v \leftarrow \text{AFUPVAR}(M(x), I, A^*(\alpha, y))$;
 if $v = 0$ then $\{ L \leftarrow L_0$; return $\}$;
 if $v = 1$ then $\{ L \leftarrow \text{concat}(L_0, ((a, b)))$; return $\}$.
3. [Bisect.] $c \leftarrow (a + b)/2$;
 $A_1(\alpha, y) \leftarrow \text{IPBHTMV}(2, A(\alpha, y), -1)$; $A_2(\alpha, y) \leftarrow \text{IPTR1}(2, A(\alpha, y), 2)$.
4. [Left recursive call.]
 $L_1 \leftarrow \text{AFPRINCS}(M(x), I, A_1(\alpha, y), (a, c))$.
5. [Right recursive call.]
 $L_2 \leftarrow \text{AFPRINCS}(M(x), I, A_2(\alpha, y), (c, b))$.
6. [Combine.] $L \leftarrow \text{concat}(L_1, L_0, L_2)$ ■

Figure 53: **AFPRINCS** Algebraic Field Polynomial Root Isolation, Normalized Coefficient Sign Variation Method

was 3, and 162 bisections were needed for the sign computations. The average time for the three root isolations was 177 ms. In contrast, the height of the search tree for the call to **IPRICS** on the norm was 4 and the number of nodes was 12. Moreover, the time required to isolate the roots of the norm was 50 ms and the time to compute the norm was 83 ms. For this example the norm based algorithm has fewer nodes and a comparable computing time (superior if the norm is only computed once).

For the second example, the maximum height of the search tree of the **AFPRICS** for the three conjugate polynomials $B(\alpha, y)$ is 5 and the number of nodes for the search tree is only 24. A total of 334 bisections were required for all sign computations. The total time for **AFPRICS** to isolate the real roots of these three polynomials was approximately 1.2 seconds and the average time was 0.4 seconds. Both the height and the number of nodes are significantly smaller than the height and number of nodes in the tree to isolate the real roots of the norm; however, the times are comparable.

6.4.1 Computing Time

A bound on the maximum computing time of **AFPRICS** can be obtained in the same way the bound for **IPRICS** was obtained. The height of the tree and the number of nodes at level h are bounded using the corresponding bound for the norm (Theorem 82). The main difference is that the time for sign computation can not be ignored.

Theorem 103 (Comp. Time of AFPRICS) *Let $m = \deg(A(x))$, $n = \deg(B(\alpha, y))$, and $|A(x)|_1, |B(\alpha, y)|_1 \leq d$. Then the computing time of **AFPRICS** is dominated by $m^{11}n^6 \log(d)^3$.*

PROOF. The proof is similar to the proof of the computing time bound for the integral polynomial algorithm **IPRICS** (Theorem 53). In this case, Davenport's theorem applied to the norm implies that $hk \preceq m^2 n L(de)$, where h corresponds to a level of the search tree and k is the number of intervals that need to be bisected. Therefore, by Theorem 73, the k translations at level h require time bounded by $m^3 n^4 L(de)$. Therefore, since the height of the tree is bounded by $m^2 n L(de)$, the total time for translations is bounded by $m^5 n^5 L(de)^2$.

Since the size of the coefficients of the transformed polynomials at level h are dominated by $nh + L(e)$, Theorem 75 implies that the time to compute the coefficient sign variations of the k polynomials, of degree n , at level h is dominated by $k n m^5 (nh + L(e))^2$ which is dominated by $m^9 n^5 L(de)^2$. Therefore the total time required for sign computations is bounded by $H m^9 n^5 L(de)^2$, where H is a bound on the height of the tree. Since by Theorem 82, $H \preceq b y m^2 n L(de)$, the theorem is proved ■

6.4.2 Empirical Behavior

The computing time bound for **AFPRICS** is significantly worse than what typically occurs in practice. Table 32 reports the average computing times (in ms) and statistics for **AFPRICS** on the random polynomials described in Section 6.1. Recall that C is the number of real roots of the five random polynomials, $A(x)$, of degree m and that for each root α of $A(x)$, ten polynomials $B(\alpha, y)$ were generated and their roots were then isolated. H is the average height of the search tree, N is the average number of nodes, and N' is the average number of leaf nodes. H is defined to be the maximum of the heights of the search trees for the positive and negative roots, and N and L

are the sum of the number of nodes and leaf nodes, respectively, for the search trees required to isolate the negative and positive roots. Computing times for **AFPRICS** and a modified version called **AFPRICS2** are listed. The algorithm **AFPRICS2** is a slight modification of **AFPRICS**, where the algorithm that counts the number of sign variations stops as soon as two variations are obtained. B_1 and B_2 are the average number of bisections required for all sign computations in the algorithms **AFPRICS** and **AFPRICS2** respectively.

Table 32: Empirical Behavior of **AFPRICS** for Random Polynomials

m	n	C	r	H	N	N'	AFPRICS	B_1	AFPRICS2	B_2
5	5	7	1.80	1.97	4.20	3.10	228	39.41	225	37.20
5	10	7	2.17	2.66	6.80	4.40	614	116.37	618	109.46
5	15	9	2.44	2.81	7.44	4.72	970	154.84	940.54	143.56
5	20	5	3.04	3.34	8.96	5.48	1716	293.66	1611	268.28
10	5	10	1.94	2.22	4.90	3.45	685	75.08	666	70.41
15	5	11	1.71	2.02	4.49	3.25	796	35.64	769	33.28
20	5	16	1.81	2.11	4.70	3.35	1413	43.47	1373	40.56

6.5 Effect of the Length of the Isolating Interval

In this section we present an improvement to the coefficient sign variation algorithm **AFPRICS**. The idea is to retain the refined isolating interval for α each time a sign computation is performed. The benefit of this is that far fewer bisections are required for the sign computations. The tradeoff is that for some sign computations the size of the endpoints of the isolating interval are larger and hence the evaluations and polynomial transformations are more costly. Despite this tradeoff, the computing

times of the algorithm which retains the refined isolating intervals are smaller than the corresponding times for the algorithm which does not retain the refined isolating intervals. A further advantage of the improved algorithm is that a much better computing time bound can be proven. The reason for this is that the bound for the total number of bisections needed for sign computations during the root isolation is the same as the bound which was previously applied to each sign computation.

Theorem 104 (Comp. Time of AFPRICSIR) *Let $m = \deg(A(x))$, $n = \deg(B(\alpha, y))$, $|A(x)|_1 \leq d$, and $|B(\alpha, y)|_1 \leq e$. Then the computing time of AFPRICSIR is dominated by $m^9 n^4 L(de)^2 + m^5 n^5 \log(de)^2$.*

PROOF. Recall from the proof of Theorem 103 that the time for all translations is dominated by $m^5 n^5 L(de)^2$. The time for all sign computations is bounded by the cost for one sign computation whose coefficients are bounded by $m^2 n^2 L(de)$, the bound for the coefficients of the transformed polynomials. By Theorem 75 this is bounded by $m^5 (m^2 n^2 L(de))^2$ ■

Table 33 reports the computing times (in ms) and statistics of the improved algorithm **AFPRICSIR** (Algebraic Field Polynomial Root Isolation using the Coefficient Sign variation method, Interval Refinement). The times and number of bisections should be compared to those in Table 32.

6.6 Use of Interval Arithmetic

In this section we present a version of the coefficient sign variation method that uses interval arithmetic. The algorithm takes as input a polynomial whose coefficients

Table 33: Empirical Behavior of AFPRICSIR for Random Polynomials

m	n	C	r	H	N	N'	AFPRICSIR	B_1
5	5	7	1.80	1.44	4.20	3.10	191	4.51
5	10	7	2.17	2.66	6.80	4.40	521	6.27
5	15	9	2.44	2.36	7.44	4.72	818	6.06
5	20	5	3.04	2.68	8.96	5.48	1379	7.68
10	5	10	1.94	1.66	4.90	3.45	497	5.45
15	5	11	1.71	1.69	4.49	3.25	700	4.23
20	5	16	1.81	1.74	4.70	3.35	1215	4.51

are binary rational intervals. Such a polynomial is called an interval polynomial. It is assumed that the leading coefficient is an interval that does not contain zero. Because of this assumption, the degree is well defined. The algorithm proceeds in exactly the same way as any of the other sign variation algorithms except that interval arithmetic is used in the various polynomial transformations. Furthermore, the sign of an interval may not be well defined. If an interval does not contain zero then the sign of the interval is the sign of any number in the interval. If zero is contained in the interval, the sign remains undetermined. If, when counting the number of coefficient sign variations, an undetermined sign is encountered, then the algorithm returns the corresponding interval, which has an undetermined number of real roots.

This algorithm can be used to obtain information concerning the real roots of a polynomial with real algebraic number coefficients. In many cases it provides isolating intervals for the algebraic polynomial. In fact, if the algorithm terminates without encountering any undetermined signs, then isolating intervals have been obtained for

any polynomial contained in the interval polynomial. A polynomial is contained in an interval polynomial if its coefficients are contained in the corresponding interval coefficients. In general an interval polynomial may not have a well defined number of real roots. That is there may be two polynomials contained in the interval polynomial that have different numbers of real roots. However, if the interval polynomial is obtained from a squarefree algebraic polynomial, the interval coefficients can be made small enough so that the number of roots is well defined. This follows from the continuity of the roots as a function of the coefficients. For a proof of this theorem see the book by Marden [37]. The problem of how small the intervals have to be before the number of roots is well defined is investigated in Chapter VII.

Given a polynomial $B(\alpha, y)$, where $A(\alpha) = 0$ and $\alpha \in I$, we first obtain an interval polynomial $\overline{B}(y)$ whose coefficients contain the coefficients of $B(\alpha, y)$. The width of the coefficients of the interval polynomial can be made as small as desired by refining the isolating interval I . We use the algorithm **AFPTINP** from Section 5.5 for computing the interval polynomial. Recall, that **AFPTINP** uses interval arithmetic to bound the range of $b_j(I)$ for each of the coefficients of $B(\alpha, y)$. The range of $b_j(I)$ is bounded using the mean value theorem method described in Section 5.1.4. If the width of the resulting bound for $b_j(I)$ is not less than the desired width, then the isolating interval I is bisected and the range is bounded again. The version of **AFPTINP** that we use retains the refined isolating interval as it continues with the remaining coefficients.

After the corresponding interval polynomial has been obtained, the algorithm

BRIPRICS (Binary Rational Interval Polynomial Root Isolation using the Coefficient Sign variation method) is used to attempt to isolate the roots. **BRIPRICS** begins by computing a root bound with the algorithm **BRIPRB** which provides a root bound for any polynomial contained in the interval polynomial. **BRIPRB** uses the same approach as the algorithm **AFPIRB** in Section 5.5; however, since interval bounds are already known, they do not have to be recomputed. A root bound can be computed since the leading coefficient does not contain zero. **BRIPRICS** proceeds in the same way as **IPRICS** except interval arithmetic is used for the polynomial transformations and the number of coefficient sign variations may be undefined if an interval sign can not be determined. If **BRIPRICS** does not encounter any undetermined signs, then we have obtained isolating intervals for the algebraic polynomial $B(\alpha, y)$. If an undetermined sign was obtained, there are several ways to proceed. We can always revert to the exact algorithm **AFPRICS**. Alternatively, we can further refine I so that the widths of the interval coefficients are smaller and try again. Instead of starting from scratch, we can attempt to isolate the roots of the transformed polynomials corresponding to the intervals that produced the undetermined signs.

In practice we have observed, for random polynomials, that even for polynomials with fairly wide intervals, undetermined signs are rarely encountered. In fact for the 670 random polynomials described in Section 6.1 only 29 failures due to undetermined signs were obtained with intervals having widths as large as one. With widths equal to 2^{-5} only two undetermined signs were encountered, and with widths equal to 2^{-10} no undetermined signs were encountered.

Consider the first example polynomial $B(\alpha, y)$ from Section 6.1. For $\alpha \in (-4, -3)$, this polynomial is contained in the interval polynomial

$$\begin{aligned}\overline{B}(y) = & (-9905.72903-, -9905.72829+)x^5 + (-12034.23322-, -12034.23240+)x^4 + \\ & (-8494.18660-, -8494.18597-)x^3 + (-9224.58137-, -9224.58073+)x^2 + \\ & (651.69435+, 651.69437+)x + (2009.85695-, 2009.85712+)\end{aligned}$$

BRIPRCS was able to determine that this polynomial has three real roots. The search tree associated by **BRIPRCS** has height equal to 3 and 6 nodes. The reason this is different than the search tree associated with **AFPRICS** is that a slightly larger root bound was used. The following trace, for the positive roots, illustrates the algorithm. $B(y)$ is the transformed polynomial associated with the preceding node, and $B^*(y)$ is the polynomial whose positive roots correspond to the roots of $B(y)$ in the interval $(0, 1)$. By Descartes' rule of signs, $\text{var}(B^*(y))$ exceeds the number of positive roots, and hence the number of roots of $B(y)$ in the interval $(0, 1)$, by an even number.

Node = (0, 0)

$$\begin{aligned}B(y) = & (10143465.77116+, 10143466.52204+)y^5 + \\ & (-3080763.70406+, -3080763.49488-)y^4 + \\ & (543627.90183-, 543627.94225+)y^3(-147593.30191+, -147593.29175+)y^2 + \\ & (-2606.77750-, -2606.77740+)y + (2009.85695-, 2009.85712+)\end{aligned}$$

$$B^*(y) = (2009.85695-, 2009.85712+)y^5 + (7442.50724+, 7442.50820-)y^4 +$$

$$(-137921.84243-, -137921.83015+)y^3 + (105305.90058-, 105305.97379+) y^2 + (-2436665.63140+, -2436665.30963-)y + (7458139.74647-, 7458140.75739-)$$

$$\text{var}(B^*) = 4$$

$$\text{Node} = (1, 0)$$

$$B(y) = (316983.30535-, 316983.32881+)y^5 + (-192547.73150+, -192547.71843-)y^4 + (67953.48773-, 67953.49278+)y^3 + (-36898.32548-, -36898.32294-)y^2 + (-1303.38875-, -1303.38870+)y + (2009.85695-, 2009.85712+)$$

$$B^*(y) = (2009.85695-, 2009.85712+)y^5 + (8745.89599+, 8745.89690+)y^4 + (-22013.31100-, -22013.30654-)y^3 + (-30463.25172+, -30463.23703+)y^2 + (-162500.00274-, -162499.97088+)y + (156197.20429+, 156197.24865-)$$

$$\text{var}(B^*) = 2$$

$$\text{Node} = (2, 0)$$

$$B(y) = (9905.72829+, 9905.72903-)y^5 + (-12034.23322-, -12034.23240+)y^4 + (8494.18597-, 8494.18660-)y^3 + (-9224.58137-, -9224.58073+)y^2 +$$

$$(-651.69437+, -651.69435+)y + (2009.85695-, 2009.85712+)$$

$$\begin{aligned} B^*(y) = & (2009.85695-, 2009.85712+)y^5 + (9397.59037-, 9397.59125+)y^4 + \\ & (8267.21061+, 8267.21307-)y^3 + (-2991.15491+, -2991.15050+)y^2 + \\ & (-15277.09815+, -15277.09321-)y + (-1500.73776-, -1500.73474+) \end{aligned}$$

$$\text{var}(B^*) = 1$$

$$\text{Node} = (2, 1)$$

$$\begin{aligned} B(y) = & (9905.72829+, 9905.72903-)y^5 + (37494.40824+, 37494.41273-)y^4 + \\ & (59414.53601+, 59414.54724+)y^3 + (43109.86014-, 43109.87490+)y^2 + \\ & (7773.40937-, 7773.41949+)y + (-1500.73776-, -1500.73474+) \end{aligned}$$

$$\begin{aligned} B^*(y) = & (-1500.73776-, -1500.73474+)y^5 + (269.72058+, 269.74578-)y^4 + \\ & (59196.12004+, 59196.20544+)y^3 + (220377.19506+, 220377.34148-)y^2 + \\ & (309243.00937-, 309243.13618-)y + (156197.20429+, 156197.24865-) \end{aligned}$$

$$\text{var}(B^*) = 1$$

$$\text{Node} = (1, 1)$$

$$\begin{aligned}
B(y) = & (316983.30535-, 316983.32881+)y^5 + \\
& (1392368.79524+, 1392368.92564-)y^4 + \\
& (2467595.61520+, 2467595.90720+)y^3 + \\
& (2181508.80217+, 2181509.13297-)y^2 + (943486.02421-, 943486.21412-)y + \\
& (156197.20429+, 156197.24865-)y
\end{aligned}$$

$$\begin{aligned}
B^*(y) = & (156197.20429+, 156197.24865-)y^5 + \\
& (1724472.04568-, 1724472.45736-)y^4 + \\
& (7517424.94194+, 7517426.47592-)y^3 + (16235010.20991-, 16235013.07729-)y^2 + \\
& (17427016.55046+, 17427019.23865+)y + (7458139.74647-, 7458140.75739-)
\end{aligned}$$

$$\text{var}(B^*) = 0$$

We conclude this section with the average computing times (in ms) for converting the random polynomial from Section 6.1 to interval polynomials (using **AFPTINP**) and isolating their real roots (using **BRIPRICS**). We executed these algorithms three times for each of the random algebraic polynomials. In the first case the width of the interval coefficients was set to be 2^{-10} , in the second case the width was 2^{-5} and in the third case the width was 1. The times are listed in Table 34. “Fail” is the number of times the algorithm failed. No failures occurred for interval polynomials of width 2^{-10} , two failures occurred for interval polynomials of width 2^{-5} , and 27 failures occurred for polynomials of width 1.

The computing times for the interval algorithm are comparable but more costly

than **AFPRICS**; however, we are using exact interval arithmetic. The times decrease as the interval widths become larger. Moreover, the time for conversion to an interval polynomial is more costly when the degree of the defining polynomial for α is large, while the time for **BRIPRICS** only increase slightly as the degree of α increases. However, the cost of **BRIPRICS** becomes more costly than **AFPTINP** as the degree of $B(\alpha, y)$ increases.

It is likely that if instead we were to use floating point interval arithmetic the times could be significantly improved. However, it is possible that more failures will occur since the endpoints must be rounded.

Table 34: Empirical Behavior of **BRIPRICS** for Random Polynomials

m	C	n	r	W	Fail	AFPTNP	BRIPRICS
5	7	5	1.80	2^{-10}	0	310	150
5	7	10	2.17	2^{-10}	0	413	757
5	9	15	2.44	2^{-10}	0	545	1841
5	5	20	3.04	2^{-10}	0	662	4313
10	10	5	1.94	2^{-10}	0	765	213
15	11	5	1.71	2^{-10}	0	1121	249
20	16	5	1.81	2^{-10}	0	2027	317
5	7	5	1.80	2^{-5}	0	230	135
5	7	10	2.17	2^{-5}	1	315	739
5	9	15	2.44	2^{-5}	0	440	1956
5	5	20	3.04	2^{-5}	1	540	4347
10	10	5	1.94	2^{-5}	0	609	230
15	11	5	1.71	2^{-5}	0	883	204
20	16	5	1.81	2^{-5}	0	1647	272
5	7	5	1.81	1	2	171	142
5	7	10	2.19	1	7	242	807
5	9	15	2.43	1	6	344	2221
5	5	20	3.04	1	5	427	3528
10	10	5	1.96	1	3	467	181
15	11	5	1.71	1	0	666	196
20	16	5	1.83	1	3	1306	252

CHAPTER VII

Real Root Isolation in Cylindrical Algebraic Decomposition

Real root isolation is an integral and time consuming part of cylindrical algebraic decomposition based quantifier elimination. This is our main application of real root isolation. Root Isolation can be used to obtain a one-dimensional CAD and is used in the extension phase, where a d -dimensional CAD is extended to a $d + 1$ -dimensional CAD. The number of cells in a stack over a cell in a d -dimensional CAD is determined by substituting a sample point for the cell into the $d + 1$ -dimensional polynomials and isolating their real roots. Since we need to know how many real roots there are, we need a real root isolation algorithm instead of a numeric algorithm for approximating the roots.

The following example illustrates how real root isolation is used in CAD. For this example we use the circle $C(x, y) = x^2 + y^2 - 4$ and the hyperbola $H(x, y) = xy - 1$. The projection consists of x , the leading coefficient of $H(x, y)$, $D(x) = \text{disc}(C(x, y)) = -4(x^2 - 4)$, and $R(x) = \text{res}(C(x, y), H(x, y)) = x^4 - 4x^2 - 4$. The real roots of these projection polynomials correspond to a vertical asymptote and to the projection of the points of intersection and tangency. The number of cells in the one-dimensional CAD is determined by isolating the real roots of the univariate projection

polynomials. $R(x)$ has four real roots approximately equal to -1.93 , -0.52 , 5.2 , and 1.93 . These real algebraic numbers are represented by the minimal polynomial $R(x)$ and the isolating intervals $(-2, -1)$, $(-1, 0)$, $(0, 1)$, and $(1, 2)$. Since the three projection polynomials have a total of 7 distinct real roots, there are 15 cells in the one-dimensional CAD.

In the extension phase sample points for these 15 cells are substituted into the bivariate polynomials $H(x, y)$ and $C(x, y)$ to determine how many cells there are in each cylinder over the cells in the induced one-dimensional CAD. Before, isolating the real roots, a squarefree basis is computed. For example, let α be the root of $R(x)$ in the interval $(1, 2)$. Since, $\gcd(C(\alpha, y), H(\alpha, y)) = y - 1/\alpha$, a squarefree basis is $y - 1/\alpha$ and $y + 1/\alpha$. By isolating the real roots of these basis polynomials (in this case this is trivial), we can determine that there are 5 cells in the cylinder over the one point cell containing α .

In this chapter we examine the performance of the various root isolation algorithms in the context of CAD. The first section compares the different algorithms using polynomials that arise in random two-dimensional CADs. In the following section we use CAD to study ill-conditioned polynomials. An ill-conditioned polynomial is a polynomial where a small change in its coefficients leads to a large change in its roots. In terms of real root isolation an ill-conditioned polynomial is a polynomial such that a small change in its coefficients changes the number of real roots. If a polynomial is ill-conditioned an interval polynomial containing it must have intervals with small widths, if the number of real roots is to be well defined.

7.1 Resultants, Discriminants, and Two-Dimensional CAD

In this section several experiments are performed using polynomials that arise in the CAD algorithm. The Integral polynomial root isolation algorithms are compared on random resultant and discriminants. The algebraic polynomial root isolation algorithms are compared on polynomials that are obtained by substituting the roots of discriminants and resultants into bivariate polynomials.

In the first experiment, we generated pairs of random bivariate polynomials and computed their discriminants and resultant. We then used the three different root isolation algorithms to isolate the roots of the discriminants and resultants.

In the second experiment we again generated pairs of random bivariate polynomials, $B_1(x, y)$ and $B_2(x, y)$, and computed their resultant and discriminants: $R(x) = \text{res}(B_1(x, y), B_2(x, y))$, $D_1(x) = \text{disc}(B_1(x, y))$, and $D_2(x) = \text{disc}(B_2(x, y))$. After isolating the roots of $D_i(x)$ and $R(x)$, we substituted them back into the two bivariate polynomials. and then isolated the real roots of these algebraic polynomials. Since the polynomials obtained from substituting the roots of $D_i(x)$ and $R(x)$ may not be squarefree, we first had to compute their greatest squarefree divisor. In fact, since we are substituting at the roots of the discriminant, we are guaranteed to get a polynomial that is not squarefree.

Since the computation of the greatest squarefree divisor can be costly when the polynomial is not squarefree, (we are using the monic PRS algorithm with a fast relative primality check), we used an experiment that only substituted at the roots of the resultant. In this case it is likely that the resulting polynomials will be squarefree.

Tables 35, 37, and 39 reports the computing times and statistics for random resultants. In these tables, $m = \deg_x(B_i(x, y))$, $n = \deg_y(B_i(x, y))$, and $N = \deg(\text{res}(B_1(x, y), B_2(x, y)))$. r is the average number of real roots, H is the average height, N is the average number of nodes, and L is the average number of leaf nodes for the search trees for **IPRIST** and **IRPICS**. In Table 37, S is the average number of bits in the Sturm sequence. In Table 39, D is the average number of bits in the primitive derivative sequence, r' is the average number of real roots of the derivative sequence, and B is the average number of bisections per sign computation.

Tables 36, 38, and 40 reports the computing times and statistics for random discriminants. In these tables, $m = \deg_x(B_i(x, y))$, $n = \deg_y(B_i(x, y))$, and $N = \deg(\text{disc}(B_1(x, y), B_2(x, y)))$.

Table 35: **IPRICS** Statistics for Random Resultants

m	n	k	N	r	H	N	L	IPRICS
2	2	10	8	2.66	5.08	13.22	7.61	30.9
2	3	10	12	2.96	5.46	15.46	8.73	61.2
2	4	10	16	3.18	6.45	18.84	10.42	112.0
2	5	10	20	3.54	6.40	19.26	10.63	168.5
2	6	10	24	3.46	7.03	21.10	11.55	256.8
2	7	10	28	3.90	7.58	23.74	12.87	393.2
2	8	10	32	4.16	7.54	24.24	13.12	508.0
2	9	10	36	4.18	7.63	24.56	13.28	682.4
2	10	10	40	4.22	7.76	24.74	13.37	850.3
3	3	10	18	3.54	6.38	18.62	10.31	119.0
4	4	10	32	4.12	6.92	21.64	11.82	337.8
5	5	10	50	5.04	7.88	25.48	13.74	979.0
6	6	10	72	5.38	8.05	26.74	14.37	2279.0

In the second set of experiments we study the behavior of the various real root

Table 36: **IPRICS** Statistics for Random Discriminants

m	n	k	N	r	H	N	L	IPRICS
2	2	10	4	2.06	2.98	6.88	4.44	7.7
2	3	10	8	2.64	5.76	15.20	8.60	37.4
2	4	10	12	3.08	7.16	20.80	11.40	86.6
2	5	10	16	3.30	7.09	20.66	11.33	127.5
2	6	10	20	3.58	7.35	22.60	12.30	200.2
2	7	10	24	3.82	8.26	24.86	13.43	317.9
2	8	10	28	4.56	8.70	27.82	14.91	489.3
2	9	10	32	4.40	9.10	28.40	15.20	658.4
2	10	10	36	4.30	8.50	27.88	14.94	809.8
3	3	10	12	2.96	5.57	15.68	8.84	57.4
4	4	10	24	4.00	7.76	23.40	12.70	235.4
5	5	10	40	4.48	8.21	26.18	14.09	642.2
6	6	10	60	5.00	8.79	28.82	15.41	1674.7

Table 37: **IPRIST** Statistics for Random Resultants

m	n	k	N	r	H	N	L	S	IPRIST	IPPNPRS
2	2	10	8	2.66	3.82	8.72	5.36	190.8	307.5	29.5
2	3	10	12	2.96	3.63	8.88	5.44	518.0	3505.8	115.8
2	4	10	16	3.18	4.71	11.24	6.62	1017.0	20280.2	432.9
3	3	10	12	3.54	5.01	12.22	7.11	889.4	19144.6	524.5

Table 38: **IPRIST** Statistics for Random Discriminants

m	n	k	N	r	H	N	L	S	IPRIST	IPPNPRS
2	2	10	4	2.06	2.15	4.82	3.41	32.0	9.2	3.5
2	3	10	8	2.64	4.07	9.60	5.80	200.6	330.7	32.5
2	4	10	12	3.08	5.24	12.76	7.38	544.1	3841.5	171.6
3	3	10	12	2.96	4.14	10.04	6.02	364.4	1850.6	99.5

Table 39: **IPRIDS** Statistics for Random Resultants

m	n	k	N	r	r'	B	D	IPRIDS
2	2	10	8	2.66	2.49	1.73	42.1	147.6
2	3	10	12	2.96	3.02	2.00	65.3	443.3
2	4	10	16	3.18	3.37	2.16	88.7	973.7
2	5	10	20	3.54	3.80	2.25	112.8	1879.8
2	6	10	24	3.46	4.08	2.41	137.0	3276.5
2	7	10	28	3.90	4.63	2.38	160.9	5429.6
2	8	10	32	4.16	4.97	2.41	185.6	8486.3
2	9	10	36	4.18	5.31	2.46	209.9	12410.3
2	10	10	40	4.22	5.43	2.49	234.3	17182.2
3	3	10	18	3.54	3.60	2.21	70.1	1152.0
4	4	10	32	4.12	4.59	2.44	100.7	5561.6
5	5	10	50	5.04	6.04	2.58	134.8	21784.6
6	6	10	72	5.38	7.26	2.61	172.4	67634.8

Table 40: **IPRIDS** Statistics for Random Discriminants

m	n	k	N	r	r'	B	D	IPRIDS
2	2	10	4	2.06	2.10	1.18	21.0	22.7
2	3	10	8	2.64	2.53	1.80	44.9	155.7
2	4	10	12	3.08	3.09	2.03	69.4	472.8
2	5	10	16	3.30	3.44	2.12	95.1	1034.9
2	6	10	20	3.58	3.87	2.12	121.9	1992.5
2	7	10	24	3.82	4.44	2.24	148.4	3627.6
2	8	10	28	4.56	5.21	2.19	175.4	6177.6
2	9	10	32	4.40	5.49	2.33	202.5	9489.4
2	10	10	36	4.30	5.99	2.20	230.3	14023.8
3	3	10	12	2.96	2.77	1.94	48.0	364.5
4	4	10	24	4.00	4.37	2.22	78.7	2484.9
5	5	10	40	4.48	5.34	2.43	113.6	10867.4
6	6	10	60	5.00	7.17	2.43	152.2	41778.3

isolation algorithms in the computation of a CAD for a pair of random bivariate polynomials. Two bivariate polynomials, $B_1(x, y)$ and $B_2(x, y)$ are generated and their resultant, $R(x)$, and discriminants, $D_1(x)$ and $D_2(x)$, are computed. The univariate polynomials $R(x)$, $D_1(x)$, and $D_2(x)$ are factored into irreducibles and the real roots of the irreducible factors are isolated. These isolating intervals along with the corresponding minimal polynomials define sample points for the induced CAD. These sample points are then substituted into the bivariate polynomials $B_1(x, y)$ and $B_2(x, y)$. Let α be a sample point. After substituting α into $B_i(x, y)$ the real roots of $\text{gsfd}(B_i(\alpha, y))$ are isolated. In general $B_i(\alpha, y)$ may not be squarefree. In particular, if α is a root of $\text{disc}(B_i(x, y))$, then we know that $B_i(\alpha, y)$ has a multiple root. In an actual CAD computation, a squarefree basis will be computed for $B_1(\alpha, y)$ and $B_2(\alpha, y)$, and the number of real roots of the basis polynomials determine the number of cells in the cylinder over α . In our experiment we isolate the real roots of $\text{gsfd}(B_i(\alpha, y))$ for all of the real roots, α , of $R(x)$ and $D_i(x)$.

In experiment 1, $\deg_x(B_i(x, y)) = 2$, $\deg_y(B_i(x, y)) = 4$, $\deg(R(x)) = 16$, and $\deg(D_i(x)) = 12$. All three polynomials are irreducible. $D_1(x)$ has 4 real roots contained in the intervals: $I_1 = (-4, -3)$, $I_2 = (-3, -2)$, $I_3 = (-1, 0)$, $I_4 = (0, 1/4)$. $D_2(x)$ has 6 real roots contained in the intervals: $I_1 = (-1/2, -1/4)$, $I_2 = (-1/4, 0)$, $I_3 = (0, 1/2)$, $I_4 = (1/2, 5/8)$, $I_5 = (5/8, 3/4)$, $I_6 = (1, 2)$. $R(x)$ has 6 real roots contained in the intervals: $I_1 = (-2, -1)$, $I_2 = (-1, 0)$, $I_3 = (0, 1/4)$, $I_4 = (1/4, 1/2)$, $I_5 = (1/2, 1)$, $I_6 = (4, 8)$. The number of real roots of $B_1(x, y)$ and $B_2(x, y)$ when substituted at the roots of $D_1(x)$, $D_2(x)$, and $R(x)$ are 16, 26, and 26 respectively.

The total time for **AFPRICS** was 27.2 seconds. The total time for **AFPRIST** was 4998.5 seconds The total time for **AFPRIDS** = 56.2 seconds

In experiment 2, $\deg_x(B_i(x, y)) = 3$, $\deg_y(B_i(x, y)) = 3$, $\deg(R(x)) = 18$, and $\deg(D_i(x)) = 12$. All three polynomials are irreducible. $D_1(x)$ has 2 real roots contained in the intervals: $I_1 = (-2, 0)$, $I_2 = (1, 2)$. $D_2(x)$ has 2 real roots contained in the intervals: $I_1 = (1, 2)$, $I_2 = (2, 4)$. $R(x)$ has 2 real roots contained in the intervals: $I_1 = (85/64, 171/128)$, $I_2 = (171/128, 43/32)$, The number of real roots of $B_1(x, y)$ and $B_2(x, y)$ when substituted at the roots of $D_1(x)$, $D_2(x)$, and $R(x)$ are 6, 10, and 12 respectively. The total time for **AFPRICS** was 5.4 seconds. The total time for **AFPRIDS** was 9.1 seconds. **AFPRIST** did not complete in a reasonable amount of time.

In experiment 3, $\deg_x(B_i(x, y)) = 2$, $\deg_y(B_i(x, y)) = 5$, $\deg(R(x)) = 20$, and $\deg(D_i(x)) = 16$. All three polynomials are irreducible. $D_1(x)$ has 4 real roots contained in the intervals: $I_1 = (-4, -2)$, $I_2 = (-2, 0)$, $I_3 = (0, 1/4)$, $I_4 = (1, 2)$. $D_2(x)$ has 6 real roots contained in the intervals: $I_1 = (-2, -1)$, $I_2 = (-3/4, -1/2)$, $I_3 = (0, 1)$, $I_4 = (1, 3/2)$, $I_5 = (13/4, 27/8)$, $I_6 = (27/8, 7/2)$. $R(x)$ has 4 real roots contained in the intervals: $I_1 = (-16, -8)$, $I_2 = (-8, 0)$, $I_3 = (0, 1)$, $I_4 = (1, 2)$. The number of real roots of $B_1(x, y)$ and $B_2(x, y)$ when substituted at the roots of $D_1(x)$, $D_2(x)$, and $R(x)$ are 18, 28, and 18 respectively. The total time for **AFPRICS** was 182 seconds. The total time for **AFPRIDS** was 397 seconds **AFPRIST** did not complete in a reasonable amount of time.

In the final experiment we generate 10 pairs of random polynomials. Their resul-

tant is computed, factored, and its real roots are isolated. These roots are then substituted into the bivariate polynomials and the real roots of the resulting polynomials are isolated. The average behavior of the coefficient sign variation method applied to these algebraic polynomials is then recorded. $m = \deg_x(B_i(x, y))$, $n = \deg_y(B_i(x, y))$, and k is the number of bits in the coefficients of $B_i(x, y)$. N is the degree of $\text{res}(B_1(x, y), B_2(x, y))$. r is the average number of real roots of the univariate resultants and R is the average number of real roots of the bivariate polynomials evaluated at the roots of their resultant. H is the average height of the search tree, N is the average number of nodes, L is the average number of leaf nodes of the search tree used by **AFPRICS**. S is the average number of bisections required by all of the sign computations in **AFPRICS**.

Table 41: **AFPRICS** Statistics for Random Resultants

m	n	k	N	r	R	H	N	L	S	AFPRICS
2	2	10	8	3.20	2.00	1.16	2.31	2.16	7.28	68.5
2	3	10	12	2.80	2.04	2.18	4.36	3.18	30.46	199.7
2	4	10	16	4.20	2.24	2.31	5.00	3.50	16.04	247.2
2	5	10	20	4.20	2.60	2.94	6.83	4.42	46.14	570.6
2	6	10	24	3.00	2.43	3.10	6.97	4.48	68.25	894.2
2	7	10	28	3.80	2.79	2.88	7.02	4.51	45.61	978.8
2	8	10	32	4.60	2.74	3.21	8.17	5.09	71.51	1723.8
2	9	10	36	5.40	2.98	3.03	7.68	4.84	48.22	1615.1
2	10	10	40	3.40	2.97	3.75	8.88	5.44	96.79	2950.3

7.2 Ill-Conditioned Polynomials

In the previous section we studied the behavior of the various real root isolation algorithms for polynomials with integer and real algebraic number coefficients when applied to polynomials arising in the CAD algorithm. While these polynomials are of more interest than random polynomials, they still are derived from random problems. In this section we examine some polynomials that arise in an application which has independent interest. Again we will examine polynomials arising in CAD; however, in this case, the CAD comes from an application. The application concerns ill-conditioned polynomials and has some relevance to interval arithmetic real root isolation algorithms (see Section 6.6).

An ill-conditioned polynomial is a polynomial for which a small change in its coefficients leads to a large change in its roots. This concept was introduced by Wilkinson [50], in connection with the amount of accuracy required in various algorithms for numerically approximating the roots of a polynomial. Wilkinson introduced the polynomial

$$W_{20}(x) = \prod_{i=1}^{20} (x + i),$$

and observed that a small perturbation in this polynomial leads to a polynomial with greatly differing roots. In fact, $W_{20}(x) + 2^{-23}x^{19}$ has ten real roots and five pairs of complex conjugate roots, which are approximately equal to $-1, -2, -3, -4, -5, -6, -7, -8.007, -8.917, -20.847, -10.095 \pm 0.6435i, -11.794 \pm 1.652i, -13.992 \pm 2.519i, -16.731 \pm 2.813i, -19.502 \pm 1.940i$.

This example shows that the problem of real root isolation is not well defined if the

coefficients are not known exactly, since a polynomial with interval coefficients may not have a well defined number of real roots. That is, there may be two polynomials contained in the interval polynomial with differing number of real roots. One may be interested in knowing how small the interval coefficients have to be for the number of real roots to be well defined (i.e. all polynomials contained in the interval polynomial have the same number of real roots). For Wilkinson's example we know that the interval containing the 19th coefficient has to have width smaller than 2^{-23} .

By computing the CAD for the perturbed Wilkinson polynomial, we can determine a bound on how small the interval coefficients have to be for the Wilkinson polynomial to have 20 real roots. In general if $A(x)$ is a polynomial of degree m , the cells of the CAD for the perturbed polynomial $A(x) + D(x)$, where $D(x) = \sum_{i=0}^m d_i x^i$ with d_i indeterminant and x is the last variable, have a constant number of roots. In particular, the cells in $m + 2$ space correspond to perturbed polynomials with the same number of real roots. All polynomials in the cell containing $A(x)$ (i.e. containing the point $(d_0 = \dots = d_m = 0)$, have the same number of real roots as $A(x)$, and the smallest distance to a cell with a different number of roots corresponds to the smallest perturbation that will change the number of real roots. This distance corresponds to the minimum distance from the polynomial $A(x)$ to a root of the discriminant of $A(x) + D(x)$. To see this observe that the number of roots can only change by going through a multiple root. A bound on this distance can be computed by taking the maximum distance (the maximum is taken over CADs where each perturbation variable comes first) of the induced one-dimensional CAD from the cell containing

zero.

The CAD also gives the number of real roots for all possible perturbations. This information can be obtained by counting the number of cells in the cylinders in $m+2$ -dimensional space. The Wilkinson polynomial can be used to illustrate this.

To make the computation feasible, we perturb only one coefficient at a time. Let $W_{i,k}(\delta, x) = W_i(x) + \delta x^k$. The CAD for $W_{i,k}(\delta, x)$ gives the number of real roots for all possible values of δ . We begin by perturbing the x^4 coefficient. In this example, the only projection polynomial is the discriminant

$$D(\delta) = 110592000d^5 + 3796485012d^4 - 5746849200d^3 - \\ 255986515d^2 + 864000d + 20736$$

$D(\delta)$ has five real roots, which are approximately equal to -35.77930 , -0.044779 , -0.00810 , 0.00967 , 1.49377 . Since the number of roots in the cylinders over these cells and the cells lying between any two roots of $D(x)$ are invariant, the number of real roots for the different values of δ is obtained by substituting a sample point from the cells in the one-dimensional CAD into the polynomial $W_{i,k}(\delta, x)$. After substituting and isolating the real roots, we see that there are 3, 2, 1, 2, 3, 4, 5, 4, 3, 2, and 1 roots of the perturbed polynomials for values of δ lying in these 11 cells. Furthermore, we see that the number of real roots first changes when $\delta = -0.008$. At this point the number of distinct real roots changes from five to four with one multiple root.

Figure 54 plots the roots of $W_{i,k}(\delta, x)$ for different values of δ . Initially $\delta = 0$ and there are five real roots at -1 , -2 , -3 , -4 , and -5 . As δ becomes positive, the

roots that started at -1 and -2 move together and eventually become a multiple root when $\delta = 0.00967$. As δ further increases the multiple root becomes a pair of complex conjugate roots. When $\delta = 1.49377$, the roots at -3 and -4 collide and become a multiple root, which turns into a pair of complex conjugate roots as δ increases.

In the same way we can compute the number of roots of the perturbed Wilkinson polynomial for perturbations of the different coefficients. We computed the CAD for $W_{5,k}(\delta, x)$ for $k = 4, 3, 2, 1, 0$. The roots of the discriminant are listed in Table 42. From this data we can determine that the number of real roots changes fastest when perturbing the fourth coefficient. Furthermore, from the various CADs we note that the number of roots of the perturbed polynomial changes in the same way for all coefficients except the constant coefficient, where the number of roots goes through the sequence $(1, 2, 3, 4, 5, 4, 3, 2, 1)$ as δ goes from $-\infty$ to ∞ .

Table 42: Perturbed Wilkinson Polynomials

Poly	Roots of Discriminant
$W_{5,4}(\delta, x)$	$(-35.77930, -0.044779, -0.00810, 0.00967, 1.49377)$
$W_{5,3}(\delta, x)$	$(-236.217938, -1.77353, -0.03319, 0.03700, 0.10366)$
$W_{5,2}(\delta, x)$	$(-620.45982, -0.24373, -0.16986, 0.11499, 2.16676)$
$W_{5,1}(\delta, x)$	$(-642.92708, -2.74431, -0.40208, 0.58284, 0.78360)$
$W_{5,0}(\delta, x)$	$(-3.63143, -1.41870, 1.41870, 3.63143)$

The Wilkinson polynomial is very ill-conditioned. Random polynomials are much better behaved. To illustrate this, we computed the CAD of three random univariate polynomials with the x^4 coefficient perturbed by a variable amount, δ . Each random polynomial is of degree five with uniformly distributed ten bit coefficients and has one

Figure 54: Roots of Perturbed Wilkinson Polynomial

real root. The roots of their discriminants along with information about the resulting CAD is listed in Table 43.

$$A_1(\delta, x) = 916x^5 - \delta x^4 - 592x^4 + 243x^3 + 788x^2 + 81x - 127$$

$$A_2(\delta, x) = 980x^5 + \delta x^4 + 914x^4 - 993x^3 + 3x^2 - 505x + 415$$

$$A_3(\delta, x) = 677x^5 + \delta x^4 - 926x^4 + 179x^3 - 91x^2 - 62x + 902$$

Table 43: Perturbed Random Polynomials

Poly	Roots of Discriminant	Number of Roots
$A_1(\delta, x)$	$(-699.47661, 1241.05963, 2120.81424)$	$(3, 2, 1, 2, 3, 2, 1)$
$A_2(\delta, x)$	(-437.39912)	$(3, 2, 1)$
$A_3(\delta, x)$	(-315.24079)	$(3, 2, 1)$

The CADs for these problems were computed with Hoon Hong's implementation of his partial CAD algorithm [26]. His original implementation used the Collins-Loos root isolation algorithm for polynomials with real algebraic number coefficients. This implementation was briefly discussed in Section 6.3. We also inserted a version of the coefficient sign variation method which uses rational arithmetic instead of the integer arithmetic implementation described in Section 4.2.1. Even using rational arithmetic the improvement is significant. Table 44 lists the time required to compute the CADs discussed above. Computing times are given for the total computation and just the time for real root isolation (both integral and real algebraic polynomials). Times are given for both the Collins-Loos algorithm and the coefficient sign variation method (CSV).

Table 44: Computing Times for CADs

Poly	Collins-Loos		CSV	
	Total one	Collins-Loos	Total two	CSV
$W_{5,4}(\delta, x)$	48.8	45.7	12.3	9.1
$W_{5,3}(\delta, x)$	41.3	39.0	15.3	13.1
$W_{5,2}(\delta, x)$	37.6	36.0	10.5	8.8
$W_{5,1}(\delta, x)$	22.8	20.1	9.2	6.4
$W_{5,0}(\delta, x)$	5.3	4.4	2.2	1.2
$A_1(\delta, x)$	48.2	42.2	8.5	2.5
$A_2(\delta, x)$	30.3	24.2	6.7	0.7
$A_3(\delta, x)$	21.6	15.4	6.7	0.6

BIBLIOGRAPHY

- [1] Abramowitz and Stegun. *Handbook of Mathematical Functions*. Dover Publishing, Inc., New York, ninth edition, 1970.
- [2] A. Akritas and G. E. Collins. Polynomial real root isolation using Descartes' rule of signs. In *Proceedings of SYMSAC 76*, pages 272–275. ACM, New York, 1976.
- [3] Michael Ben-Or, Dexter Kozen, and John Henry Reif. The complexity of elementary algebra and geometry. *J. Comput. System Sci.*, 32(2):251–264, 1986.
- [4] A. T. Bharucha-Reid. Random algebraic equations. In A. T. Bharucha-Reid, editor, *Probabilistic Methods in Applied Mathematics*, volume 2, pages 1–52. Academic Press, 1970.
- [5] W. S. Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors. *J. ACM*, 18(4):478–504, October 1971.
- [6] W. S. Brown. The subresultant PRS algorithm. *ACM Transactions on Mathematical Software*, 4:237–249, 1978.
- [7] W. S. Brown and J. F. Traub. On Euclid's algorithm and the theory of subresultants. *J. ACM*, 18(4):505–514, October 1971.
- [8] William Snow Burnside and Arthur William Panton. *The Theory of Equations*, volume 1. Dublin University Press, Dublin, 1899.
- [9] G. E. Collins. Subresultants and reduced polynomial remainder sequences. *J. ACM*, 14:128–142, 1967.
- [10] G. E. Collins. The calculation of multivariate polynomial resultants. *J. ACM*, 18:515–532, 1971.
- [11] G. E. Collins. Computer algebra of polynomials and rational functions. *American Mathematical Monthly*, 80(7):725–755, August-September 1973.
- [12] G. E. Collins and E. Horowitz. The minimum root separation of a polynomial. *Math. Comp.*, 28(126):589–597, April 1974.
- [13] G. E. Collins and J. R. Johnson. Quantifier elimination and the sign variation method for real root isolation. In *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, pages 264–271. ACM, 1989.

- [14] G. E. Collins and R. Loos. Polynomial real root isolation by differentiation. In *Proceedings of SYMSAC 76*, pages 15–25, New York, 1976. ACM.
- [15] G. E. Collins and R. Loos. Real zeros of polynomials. In B. Buchberger, G. E. Collins, and R. Loos, editors, *Computer Algebra*, pages 83–94. Springer-Verlag, Wien-New York, 1982.
- [16] G. E. Collins and R. G. K. Loos. SAC-2 system documentation. Technical report. In Europe available from: R. G. K. Loos, Universität Tübingen, Informatik, D-7400 Tübingen, W-Germany. In the U.S. available from: G. E. Collins, Ohio State University, Computer Science, Columbus, OH 43210, U.S.A.
- [17] G. E. Collins, M. Mignotte, and F. Winkler. Basic arithmetic. In B. Buchberger, G. E. Collins, and R. Loos, editors, *Computer Algebra*, pages 189–220. Springer-Verlag, Wien-New York, 1982.
- [18] George E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In *Lecture Notes In Computer Science*, pages 134–183. Springer-Verlag, Berlin, 1975. Vol. 33.
- [19] M. Coste and M. F. Roy. Thom's lemma, the coding of real algebraic numbers and the computation of the topology of semi-algebraic sets. *Journal of Symbolic Computation*, 5(1,2):121–129, 1988.
- [20] J. H. Davenport. Computer algebra for cylindrical algebraic decomposition. Technical report, The Royal Institute of Technology, Dept. of Numerical Analysis and Computing Science, S-100 44, Stockholm, Sweden, 1985.
- [21] Leonard Eugene Dickson. *First Course in the Theory of Equations*. John Wiley and Sons Inc., New York, 1922.
- [22] A. J. Goldstein and R. L. Graham. A Hadamard-type bound on the coefficients of a determinant of polynomials. *Siam Review*, 16:394–395, 1974.
- [23] L. E. Heindel. *Algorithms for Exact Polynomial Root Calculation*. PhD thesis, University of Wisconsin, 1970.
- [24] L. E. Heindel. Integer arithmetic algorithms for polynomial real zero determination. *J. ACM*, 18(4):533–548, October 1971.
- [25] August Hollkott. *Finite Konstruktion geordneter algebraischer Erweiterungen von geordneten Grundkörpern*. PhD thesis, University of Hamburg, 1941.
- [26] H. Hong. *Improvements in CAD-based Quantifier Elimination*. PhD thesis, The Ohio State University, 1990.
- [27] M. Kac. On the average number of real roots of a random algebraic equation. *Bulletin of the American Mathematical Society*, 49:314–320, and 938, 1943.
- [28] H. Kempfert. On sign determination in real algebraic number fields. *Numer. Math.*, 11:170–174, 1968.

- [29] D. E. Knuth. *The Art of Computer Programming (Seminumerical Algorithms)*, volume 2. Addison-Wesley, Reading, Mass., 1st edition, 1969.
- [30] L. Langemyr. *Computing the GCD of Two Polynomial over an Algebraic Number Field*. PhD thesis, The Royal Institute of Technology, Stockholm, January 1989. TRITA-NA-8804.
- [31] L. Langemyr. Algorithms for a multiple algebraic extension. In *Proc. MEGA-90*. Birkhauser, 1990.
- [32] L. Langemyr and S. McCallum. The computation of polynomial greatest common divisors over an algebraic number field. *Journal of Symbolic Computation*, 8:429–448, 1989.
- [33] R. G. K. Loos. The algorithm description language ALDES (Report). *ACM SIGSAM Bull.*, 10(1):15–39, 1976.
- [34] R. G. K. Loos. Computing in algebraic extensions. In B. Buchberger, G. E. Collins, and R. G. K. Loos, editors, *Computer Algebra, Symbolic and Algebraic Computation*, pages 173–187. Springer Verlag, Wien-New York, 1982.
- [35] R. G. K. Loos. Generalized polynomial remainder sequences. In B. Buchberger, G. E. Collins, and R. G. K. Loos, editors, *Computer Algebra, Symbolic and Algebraic Computation*, pages 115–137. Springer-Verlag, Wien-New York, 1982.
- [36] K. Mahler. An inequality for the discriminant of a polynomial. *Michigan Mathematics Journal*, 11(3):257–262, September 1964.
- [37] M. Marden. *The Geometry of Zeros of a Polynomial in a Complex Variable*, volume 3 of *Mathematics Surveys*. American Mathematics Society, Providence Rhode Island, 1949.
- [38] M. Mignotte. Some inequalities about univariate polynomials. In P. Wang, editor, *SYMSAC '81: Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation*, page 195. Association for Computing Machinery, 1981.
- [39] M. Mignotte. Some useful bounds. In B. Buchberger, G. E. Collins, and R. G. K. Loos, editors, *Computer Algebra, Symbolic and Algebraic Computation*, pages 259–263. Springer Verlag, Wien-New York, 1982.
- [40] H. Minc and M. Marcus. *A Survey of Matrix Theory and Matrix Inequalities*. Allyn and Bacon Inc., Boston, 1964.
- [41] R. E. Moore. *Methods and Applications of Interval Arithmetic*. Studies in Applied Mathematics. SIAM, 1979.
- [42] C. M. Rubald. *Algorithms for Polynomials over a Real Algebraic Number Field*. PhD thesis, University of Wisconsin, 1973.

- [43] S. M. Rump. *Isolierung der Reellen Nullstellen Algebraischer Polynome*. PhD thesis, Universität Kaiserslautern, 1976.
- [44] S. M. Rump. On the sign of a real algebraic number. In *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation*, pages 238–241, New York, 1976. ACM.
- [45] J. Schwartz and M. Sharir. On the ‘piano movers’ problem. ii. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4:298–351, 1983.
- [46] G. Szego. *Orthogonal Polynomials*, volume 23 of *American Mathematical Society Colloquim*. American Mathematical Society, New York, 1939.
- [47] B. Trager. Algebraic factoring and rational function integration. In *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation*, pages 219–226, New York, 1976. ACM.
- [48] J. V. Uspensky. *Theory of Equations*. McGraw-Hill Book Company, New York, 1948.
- [49] M. Vincent. Sur la resolution des equations numeriques. *Jour. des Mathematique Pures et Appliques*, 1:341–372, 1836.
- [50] J. H. Wilkinson. The evaluation of the zeros of ill-conditioned polynomials. parts i and ii. *Numerische Mathematik*, 1:150–166, 167–180, 1959.
- [51] Hans Zassenhaus. A real root calculus. In John Leach, editor, *Computational Problems in Abstract Algebra*, pages 383–392. Pergamon Press, Oxford, 1970.