Algorithms and Implementations
in Computational Algebraic Geometry

by

Jeff Sommars
B.S. Wheaton College 2011
M.A.T. Wheaton College 2013

Thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mathematics
in the Graduate College of the
University of Illinois at Chicago, 2018

Chicago, Illinois

Defense Committee:
Jan Verschelde, Chair and Advisor
Gerard Awanou
Marshall Hampton
Lev Reyzin
Kevin Tucker
To my wife

and my parents
ACKNOWLEDGMENTS

First and foremost, I am grateful to my advisor Jan Verschelde for providing guidance and encouragement throughout my graduate education. No matter how busy he was, his door was always open to me. I also thank the members of my thesis committee for their helpful comments and suggestions. I am grateful to the National Science Foundation for supporting some of the research contained in this thesis, under Grant Nos. DMS 1321794 and ACI 1440534.

I am indebted to many great teachers of mathematics — especially to Andy Samide from Wheaton North High School who always challenged his students to think — and to Stephen Lovett and Terry Perciante from Wheaton College.

As a graduate student, I have been lucky enough to collaborate with a wide group of mathematicians. I thank all of my coauthors that I had the pleasure to work with: Carlos Améndola, Nate Bliss, Timothy Duff, Cvetelina Hill, Paul Görlach, Anders Jensen, David Kahle, Kathlén Kohn, Sara Lamboglia, Kisun Lee, Anton Leykin, Diane Maclagan, Chris O’Neill, Yue Ren, Ben Smith, Paolo Tripoli, Jan Verschelde, Xiangcheng Yu and Magdalena Zajaczkowska.

I have had the great fortune of traveling widely to do mathematics. I thank the Macaulay2 group for running many excellent conferences where I was able to meet brilliant mathematicians. I thank Anders Jensen for hosting me twice at Aarhus University and for patiently talking about tropical prevarieties with me. I thank Bernd Sturmfels for hosting me for a research visit at MPI Leipzig. I thank the 2016 AMS Mathematics Research Community in algebraic statistics where I was able to learn about a new field of mathematics while surrounded by mountains.
Finally, I thank the Mittag-Leffler Institute for inviting me to participate in their program on tropical geometry.

While in graduate school, I had the pleasure of working in the research department of First Trust Portfolios as a software developer. I thank Dan Godoy for mentoring me and teaching me how to be a proficient programmer. I also thank several other coworkers for their eagerness to teach me about software developement and finance: Chris Bush, Graham Homberg, Jay Huber, David McCammond-Watts, and Jared Wollen. On the rare occasions I was at UIC, I always enjoyed spending time with my officemates: Nate Bliss, Sarthak Chimni, Landon Kavlie, and Joel Stapleton.

Finally, I would like to thank my wife, my parents, and my siblings. I would never have gotten here without you.
CONTRIBUTION OF AUTHORS

Chapter 1 introduces necessary terminology and presents the problem statements I will answer throughout the work. Chapters 2 and 3 represent a major revision of two papers (77, 53), where I worked with Anders Jensen and my advisor Jan Verschelde. For this work, I wrote the code and performed the experiments, but the writing of the papers was done jointly. Chapter 4 describes a collaboration in which we introduce and implement a new monodromy method for solving polynomial systems (27). I excluded the portions of the paper that I did not heavily contribute to – in particular, section four where we perform a statistical analysis of the expected number of path tracks. Finally, Chapter 5 introduces a new software package that interfaces R to Macaulay2 (58). All three authors contributed equal amounts to this work; however, I excluded a description of the parser as I did not write as much of it as my coauthors did.
# Table of Contents

## Chapter 1: Introduction

1.1 Polynomial System Solving ........................................ 2
1.2 Algebraic Statistics .............................................. 4
1.3 Tropical Geometry ................................................. 5
1.4 Related Work ......................................................... 9
1.4.1 Tropical Prevarieties .......................................... 10
1.4.2 Monodromy ...................................................... 10
1.4.3 Computer Algebra in \( \mathbb{R} \) ................................ 11
1.5 Thesis Contribution ................................................ 11

## Chapter 2: A Few Polyhedral Techniques

2.1 Problem Statement .................................................. 13
2.2 An Enumeration Tree .............................................. 13
2.3 Closed Polyhedral Cones ......................................... 15
2.4 Half Open Polyhedral Cones ..................................... 16
2.4.1 Constructing Half Open Cones ................................... 17
2.4.2 Representation as a Closed Cone ................................ 20
2.5 Relation Tables ...................................................... 20
2.6 Traversing the Edge Skeleton of a Newton Polytope .......... 24

## Chapter 3: Algorithms for Computing Tropical Prevarieties

3.1 Algorithm 1: The Combinatorial Algorithm .................... 28
3.2 Algorithm 2: Traversing the Newton Polytopes ............... 29
3.2.1 Duplicate Cones ............................................... 30
3.2.2 Analysis of Computational Complexity ..................... 31
3.3 Algorithm 3: Dynamic Enumeration ................................ 33
3.3.1 Greedy Selection ............................................... 34
3.4 DynamicPrevariety .................................................. 37
3.4.1 Parallel Implementation ....................................... 37
3.4.2 Experimental Results ......................................... 43
3.4.2.1 \( n \)-body and \( n \)-vortex Problems ....................... 43
3.4.2.2 Cyclic-\( n \) Roots ........................................... 45
3.4.2.3 \( 4 \times 4 \) Minors of a \( 5 \times 5 \) Matrix ..................... 48
3.4.2.4 Tropicalization of del Pezzo Surfaces ................... 48
3.4.3 Using DynamicPrevariety ..................................... 49
3.5 Future Directions .................................................... 52
# TABLE OF CONTENTS (Continued)

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>SOLVING POLYNOMIAL SYSTEMS WITH MONODROMY</td>
</tr>
<tr>
<td>4.1</td>
<td>Problem Statement</td>
</tr>
<tr>
<td>4.2</td>
<td>Background and Framework Overview</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Monodromy</td>
</tr>
<tr>
<td>4.3</td>
<td>Graph of Homotopies</td>
</tr>
<tr>
<td>4.4</td>
<td>Strategies, Algorithms and Potential Functions</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Naive Dynamic Strategy</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Static Graph Strategy</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Incremental Dynamic Graph Strategy</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Specializing the Algorithms</td>
</tr>
<tr>
<td>4.4.4.1</td>
<td>Basic Static Graphs</td>
</tr>
<tr>
<td>4.4.4.2</td>
<td>Stopping Criterions</td>
</tr>
<tr>
<td>4.4.4.3</td>
<td>Edge Selection Strategy</td>
</tr>
<tr>
<td>4.5</td>
<td>Implementation</td>
</tr>
<tr>
<td>4.6</td>
<td>Experiments</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Sparse Polynomial System</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Success Rate</td>
</tr>
<tr>
<td>4.6.3</td>
<td>Comparison to Other Solvers</td>
</tr>
<tr>
<td>4.7</td>
<td>Parallel Monodromy</td>
</tr>
<tr>
<td>4.8</td>
<td>Future Direction</td>
</tr>
<tr>
<td>5</td>
<td>A COMPUTER ALGEBRA SYSTEM FOR R</td>
</tr>
<tr>
<td>5.1</td>
<td>Problem Statement</td>
</tr>
<tr>
<td>5.2</td>
<td>Basic Usage</td>
</tr>
<tr>
<td>5.2.1</td>
<td>m2r Basics</td>
</tr>
<tr>
<td>5.2.2</td>
<td>m2r and Computational Algebraic Geometry</td>
</tr>
<tr>
<td>5.2.3</td>
<td>More Examples of Macaulay2 Functionality</td>
</tr>
<tr>
<td>5.3</td>
<td>Design Philosophy</td>
</tr>
<tr>
<td>5.3.1</td>
<td>m2r Data Structures</td>
</tr>
<tr>
<td>5.4</td>
<td>The m2r Cloud</td>
</tr>
<tr>
<td>5.4.1</td>
<td>A Socket Connection Between R and Macaulay2</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Macaulay2 in the Cloud</td>
</tr>
<tr>
<td>5.5</td>
<td>Future Directions</td>
</tr>
</tbody>
</table>

| APPENDIX | 97 |
| CITED LITERATURE | 105 |
| VITA | 114 |
## LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Comparison of definitional algorithm to Algorithm</td>
</tr>
<tr>
<td>II</td>
<td>$n$-body problem timings</td>
</tr>
<tr>
<td>III</td>
<td>$n$-vortex problem timings</td>
</tr>
<tr>
<td>IV</td>
<td>cyclic-$n$ roots timings</td>
</tr>
<tr>
<td>V</td>
<td>Maximal cones of cyclic-$16$</td>
</tr>
<tr>
<td>VI</td>
<td>Cyclic-$7$ flowerGraph results</td>
</tr>
<tr>
<td>VII</td>
<td>Cyclic-$7$ completeGraph results</td>
</tr>
<tr>
<td>VIII</td>
<td>Katsura-$(n - 1)$ flowerGraph results</td>
</tr>
<tr>
<td>IX</td>
<td>Katsura-$(n - 1)$ completeGraph results</td>
</tr>
<tr>
<td>X</td>
<td>MonodromySolver timings when root count &lt; mixed volume</td>
</tr>
<tr>
<td>XI</td>
<td>MonodromySolver timings when root count = mixed volume</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Support sets and Newton polytopes</td>
</tr>
<tr>
<td>2</td>
<td>Tropical hypersurfaces</td>
</tr>
<tr>
<td>3</td>
<td>Tropical prevariety</td>
</tr>
<tr>
<td>4</td>
<td>Closed cones</td>
</tr>
<tr>
<td>5</td>
<td>Half open cones</td>
</tr>
<tr>
<td>6</td>
<td>Oriented cube</td>
</tr>
<tr>
<td>7</td>
<td>Intersection of relation tables</td>
</tr>
<tr>
<td>8</td>
<td>Cones and relation tables</td>
</tr>
<tr>
<td>9</td>
<td>Relation table intersection</td>
</tr>
<tr>
<td>10</td>
<td>Traversing edge skeleton</td>
</tr>
<tr>
<td>11</td>
<td>Difference between static and dynamic enumeration</td>
</tr>
<tr>
<td>12</td>
<td>Queue of subqueues</td>
</tr>
<tr>
<td>13</td>
<td>Dynamic decomposition</td>
</tr>
<tr>
<td>14</td>
<td>Graph of homotopies at instantiation</td>
</tr>
<tr>
<td>15</td>
<td>Graph of homotopies at termination</td>
</tr>
<tr>
<td>16</td>
<td>Sample layouts for graphs of homotopies</td>
</tr>
<tr>
<td>17</td>
<td>Socket connection process</td>
</tr>
<tr>
<td>18</td>
<td>Messages passed through socket</td>
</tr>
<tr>
<td>19</td>
<td>R connects to remote Macaulay2</td>
</tr>
</tbody>
</table>
### LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>Variables $(x_1, x_2, \ldots, x_n)$</td>
</tr>
<tr>
<td>$f(x)$</td>
<td>Polynomial</td>
</tr>
<tr>
<td>$\text{supp}(f)$</td>
<td>Support of a polynomial</td>
</tr>
<tr>
<td>$F(x)$</td>
<td>Polynomial system</td>
</tr>
<tr>
<td>$\mathbb{C}^*$</td>
<td>$\mathbb{C} \setminus {0}$</td>
</tr>
<tr>
<td>$P$</td>
<td>Polytope</td>
</tr>
<tr>
<td>$\text{NP}(f)$</td>
<td>Newton polytope of $f(x)$</td>
</tr>
<tr>
<td>$\text{dim}(P)$</td>
<td>Dimension of $P$</td>
</tr>
<tr>
<td>$C$</td>
<td>Polyhedral cone</td>
</tr>
<tr>
<td>$F$</td>
<td>Normal fan</td>
</tr>
<tr>
<td>$\mathcal{T}(f)$</td>
<td>Tropical hypersurface of $f(x)$</td>
</tr>
<tr>
<td>$\mathcal{T}(P)$</td>
<td>Tropical hypersurface of every $f(x)$ with $\text{NP}(f) = P$</td>
</tr>
<tr>
<td>$</td>
<td>\mathcal{T}(P)</td>
</tr>
<tr>
<td>$\text{in}_w(f)$</td>
<td>Initial form of $f(x)$ with respect to $w$</td>
</tr>
<tr>
<td>$F_i \wedge F_j$</td>
<td>Common refinement of two normal fans</td>
</tr>
<tr>
<td>$\mathcal{T}(F)$</td>
<td>Tropical prevariety of $F(x)$</td>
</tr>
</tbody>
</table>
LIST OF ABBREVIATIONS (Continued)

\( C_{i,j} \) \hspace{1cm} j^{th} \text{ cone of } \mathcal{T}(f_i) \\
\( G(P, C) \) \hspace{1cm} \text{Pretropism graph of polytope } P \text{ defined by } C \\
\|G(P, C)\| \hspace{1cm} \# \text{ of edges in } G(P, C) \\
\text{GPL} \hspace{1cm} \text{GNU General Public License} \\
\text{PPL} \hspace{1cm} \text{Parma Polyhedra Library} \\
\mathbf{F}(x)_p \hspace{1cm} \text{Polynomial system with parametric coefficients} \\
B \hspace{1cm} \text{Base space} \\
\mathcal{G} \hspace{1cm} \text{Graph of homotopies} \\
E(\mathcal{G}) \hspace{1cm} \text{Edge set of } \mathcal{G} \\
N(\mathcal{G}) \hspace{1cm} \text{Node set of } \mathcal{G} \\
\text{AWS} \hspace{1cm} \text{Amazon Web Services} \\
\text{EC2} \hspace{1cm} \text{Elastic Compute Cloud}
SUMMARY

In this thesis, we explore several areas of computational algebraic geometry, and develop new algorithms and software in each. We are generally interested in solving polynomial systems and applications that require solving polynomial systems.

Following the introduction, Chapters 2 and 3 develop new tools for the computation of tropical prevarieties. A tropical prevariety is a generalization of a mixed volume that has applications in tropical geometry as well as polynomial system solving. In traditional tropical geometry, a tropical prevariety is needed as the first step in computing a tropical variety (17). Tropical prevarieties are useful in polynomial system solving as they are used in generalizing polyhedral homotopies from isolated solutions to positive dimensional systems (83, 1, 52, 13, 12). Unlike in the mixed volume computation, the polytopes in the tropical prevariety computation have not been generically lifted, which makes the computation more difficult. In Chapter 3, we present experimental results using our software, including the first computation of the tropical prevariety of the cyclic-16 roots problem and the tropical prevariety required for the disproof of the conjecture by Ren, Shaw, and Sturmfels in (75).

In Chapter 4, we introduce a new framework for finding all solutions of a generic polynomial system with parametric coefficients. This framework establishes a way of describing and analyzing monodromy solvers in terms of labeled graphs. In (27), we showed that the expected number of path tracks by an algorithm that follows this framework is linear in the size of the solution set. We discuss various ways of evaluating the potential of an edge in these homotopy
SUMMARY (Continued)

graphs to find a new solution at a node. Unlike standard homotopy continuation methods, this algorithm is not pleasingly parallel; we briefly touch on an algorithm for improving its efficiency when run in parallel. This chapter also contains experimental results, showing that our high-level implementation is competitive with the existing state-of-the-art methods implemented in other software packages for standard sparse systems, and that it beats other solvers when the number of solutions is much smaller than the mixed volume.

The thesis concludes with a chapter on m2r, a software package we have developed that connects R (74) to Macaulay2 (41). The field of algebraic statistics uses algebra to provide insight into statistical problems. However, there is no dedicated software system that can provide for the dual statistical and algebraic software needs of this community. Until recently, practitioners had performed computations in Macaulay2 and then manually pulled them into R for further experimentation. We introduce m2r as the tool that solves this problem through connecting the two software packages with a persistent socket connection. We discuss basic usage of m2r and demonstrate the use of m2r on a cloud server.
CHAPTER 1

INTRODUCTION

The fundamental difficulty in many problems in mathematics is finding the solutions to a system of polynomial equations. The field of computational algebraic geometry revolves around just that challenge: finding symbolic or numeric solutions to polynomial systems through algorithmic methods. The rise of computational algebraic geometry has been enabled by the rise of the computer; without sufficient computing power, many of the algorithms would be computationally infeasible for even the most trivial of problems.

This thesis creates and implements a variety of algorithms from computational algebraic geometry. In particular, the following three problems are investigated:

1. Given a tuple of Newton polytopes \((P_1, P_2, \ldots, P_N)\) with normal fans \((F_1, F_2, \ldots, F_N)\), efficiently compute the tropical prevariety with a parallel algorithm.

2. Find all solutions of a generic polynomial system \(F(x)_p\) in a family of polynomial systems with parametric coefficients.

3. Create an R package that connects R to Macaulay2 through a persistent back-end socket connection.

The content of this thesis is based on several papers that have either been published or are currently under review \([76, 77, 53, 27, 14, 58]\). Chapters 2 and 3 contain major revisions of \([76, 77, 53]\), while Chapters 4 and 5 contain abridged versions of the original papers.
1.1 Polynomial System Solving

A polynomial $f(x)$ in $n$ variables $x = (x_1, x_2, \ldots, x_n)$ is defined to be

$$f(x) = \sum_{a \in \text{supp}(f)} c_a x^a$$

where $a = (a_1, a_2, \ldots, a_n), a_i \in \mathbb{N}^*$, and $c_a \in \mathbb{C}^*$. The finite set $\text{supp}(f)$ of exponent vectors of $f$ is called the support of the polynomial. A polynomial system $F(x)$ is a finite set of polynomials.

Homotopy continuation is a standard numerical technique for finding approximations of all solutions of polynomial systems \([72, 79]\). Many strategies for using homotopy continuation exist, including total degree homotopies, polyhedral homotopies \([85, 50]\) and regeneration \([48]\). We begin by describing homotopy continuation at a high level and abstracting away from differences in methodologies. We focus only on polynomial systems of $n$ polynomials in $n$ variables; we will assume that the polynomial systems of interest have zero dimensional solution sets.

The main idea behind numerically solving a polynomial system is that we first solve a simple polynomial system $G(x)$ and transform the solutions into the system of interest $F(x)$ using the following homotopy

$$H(x, t) = (1 - t)G(x) + tF(x) \in \mathbb{C}[x]^n, \ t \in [0, 1]$$

We call $G(x)$ the start system and $F(x)$ the target system. A good start system has solutions that are easily obtained, and does not have many more solutions than the target system. A simple example of a start system is the total degree start system
\[ G(x) = \begin{cases} 
  x_1^{d_1} - 1 = 0 \\
  x_2^{d_2} - 1 = 0 \\
  \ldots \\
  x_n^{d_n} - 1 = 0 
\end{cases} \]  

(1.3)

where \( d_i \) is chosen such that it is the degree of the \( i \)th equation of \( F(x) \). The solutions to this polynomial system can be written down without computation, as they are the roots of unity. \( G(x) \) has no fewer solutions than the target system, but if \( F(x) \) is sparse, it likely has many more solutions. Because of this, total degree homotopies frequently track many more solutions than is necessary.

Once a start system and its solutions have been set up, we can follow the homotopy paths from the start system \( (t = 0) \) and transform them to the target solutions \( (t = 1) \). A homotopy continuation algorithm pushes \( t \) from 0 to 1, keeping track of what the solutions are for \( H(x, t) \) for every value of \( t \). This happens in two steps:

1. Using the solution of \( t = t_i \), predict where a new solution will be for \( t = t_i + \epsilon \). This can be done using one of several different types of predictors, for example a tangent predictor or an Euler predictor.

2. Correct the predicted solution to the true solution of \( H(x, t_i + \epsilon) \). This is typically done using a multivariate variant of Newton’s method.

Once \( t = 1 \), a solution to the target system has been found. After this has been done for every solution of \( G(x) \), the target system has been solved.
The problem of finding all solutions of a polynomial system has a complexity similar to volume and counting problems that are in \#P \cite{29}. The cost is frequently expressed as being sensitive to the size of the output. The best that can be achieved is polynomial time cost per isolated solution or per positive dimensional component found. When solving polynomial systems with homotopy continuation, randomness is involved in choosing the paths to follow. Because of this, homotopy continuation methods have a probabilistic element, and different random choices will lead to different computation times.

It is natural to have concerns that homotopy continuation could suffer from issues of numerical precision. Towards this end, software has been developed to certify that a solution of a polynomial system is indeed a solution \cite{19}.

We define one additional term, a *generic polynomial system* \( F(x)_p \), which is a polynomial system with random coefficients \( p \in \mathbb{C}^n \). In Chapter \[1\], we will define a new monodromy method that uses homotopy continuation for computing all solutions for a generic polynomial system.

### 1.2 Algebraic Statistics

A novel application of polynomial system solving is in the new field of algebraic statistics. Algebraic statistics is defined broadly as the application of commutative algebra and algebraic geometry to statistical problems. It is generally understood to include applications of other mathematical fields that have substantial overlap with commutative algebra and algebraic geometry, such as combinatorics, polyhedral geometry, graph theory, and numerical algebraic geometry \cite{26, 80}. Over the past twenty years, researchers have discovered that many statistical areas can benefit from algebraic investigation, including discrete multivariate analysis,
discrete and Gaussian graphical models, statistical disclosure limitation, phylogenetics, and Bayesian statistics. Though the field is well-established and actively growing, the tools of algebraic statistics are not widely used among applied statisticians, largely due to the lack of off-the-shelf implementations of key algebraic algorithms in mainstream statistical software.

1.3 Tropical Geometry

Through the growing literature on polyhedral homotopy continuation \([50, 85, 83, 4, 2, 3, 52, 13]\), the field of tropical geometry has become tied to polynomial system solving. We now define sufficient terminology as is needed for Chapters 2 and 3. We refer the curious reader to [67] for an introduction to tropical geometry, and to [91, 24] for introductions to polyhedral geometry.\(^1\) We introduce \(F(x)\) as a simple example of a polynomial system that we will use as a running example throughout this section.

\[
F(x) = \begin{cases} 
  f_1 = x^2y^2z^2 + x^2y^2 + x^2z^2 + y^2z^2 + x^2 + y^2 + z^2 + 1 \\
  f_2 = xyz^2 + x^2y + xy^2 + x + y 
\end{cases} \quad (1.4)
\]

A convex polytope \(P\) is the convex hull of a finite set of points. The Newton polytope \(NP(f)\) of a polynomial is a convex polytope where the points are the support of \(f\) (see Figure 1 for an example).

---

\(^1\)When we work in tropical geometry, we will exclusively use the trivial valuation, as the algorithms we present in Chapter 3 can be generalized easily to nontrivial valuations through increasing the ambient dimension.
The ambient dimension of a polytope is the dimension of the space in which it rests and the affine dimension of a polytope is the dimension of its affine span. In a slight abuse of language, when we refer to the dimension of a polytope or write \( \dim(P) \), we exclusively mean the affine dimension. A face of a polytope \( P \) is the intersection of \( P \) with the boundary of a closed halfspace that contains \( P \). A face of a polytope is again a polytope. Faces with dimension zero, one, \( \dim(P) - 2 \), and \( \dim(P) - 1 \) are vertices, edges, ridges, and facets of \( P \), respectively.

A cone \( C \) in \( \mathbb{R}^n \) is a non-empty set that is closed under positive finite linear combinations. Every face of a polytope has an associated normal cone, which is a cone such that all of the rays inside of it are normal to that face. The set of all normal cones constitutes the normal fan.
$F$ of the polytope, which is a polyhedral fan in $\mathbb{R}^n$. Given two fans $F_1$ and $F_2$, their common refinement $F_1 \wedge F_2$ is defined as

$$F_1 \wedge F_2 = \{C_1 \cap C_2 \mid (C_1, C_2) \in F_1 \times F_2\}.$$  \hspace{1cm} (1.5)

As the common refinement of two fans is again a fan, the common refinement of three fans $F_1$, $F_2$, and $F_3$ may be computed as $(F_1 \wedge F_2) \wedge F_3$.

The tropical hypersurface of a polynomial $T(f)$ is the subfan of non-maximal cones of $\text{NP}(f)$ (see Figure 2 for examples). The initial form $\text{in}_w(f)$ is the sum of all terms in $f$ with lowest $w$-weight. Given that definition, a tropical hypersurface is more cleanly defined as

$$T(f) = \{w \in \mathbb{R}^n : \text{in}_w(f) \text{ is not a monomial}\}.$$  \hspace{1cm} (1.6)

where $\text{in}_w(f)$ is the initial form of $f$.

We now have the framework to define the key construct which we will work towards computing in Chapter 3, a tropical prevariety. A tropical prevariety $T(F)$ is the intersection of the tropical hypersurfaces defined by a set of $m$ polynomials of a polynomial system $F(x)$

$$T(f_1) \wedge T(f_2) \wedge \ldots \wedge T(f_m).$$  \hspace{1cm} (1.7)

In (17), the tropical prevariety was defined via the common refinement of a set of normal fans (see Figure 3 for the conclusion of the illustrative example). The nonzero vectors in the
prevariety are called *pretropisms*. The pretropisms are exactly the vectors normal to positive dimensional faces of each polytope in a tuple of Newton polytopes.

We also introduce a couple additional pieces of notation that will be convenient in coming chapters. Let $|\mathcal{T}(f)|$ be the number of codimension one cones in the tropical hypersurface. When we write $C \cap \mathcal{T}(f_i)$, we mean intersecting $C$ with each of the codimension one cones in $\mathcal{T}(f_i)$.

The tropical prevariety is a purely combinatorial object, depending only on the Newton polytopes of the polynomials in the system. The cancellation properties of the coefficients of an ideal are captured by the *tropical variety* of the polynomial ideal of the system. A tropical

Figure 2: Tropical hypersurfaces of $f_1$ and $f_2$. 
variety is the intersection of all possible tropical hypersurfaces $T(f)$ for each $f \in I$, but we will not focus on tropical varieties in this work. Tropical prevarieties are of interest in the polynomial system solving community because if the tropical prevariety has no cone of dimension $d$, then there cannot be a $d$ dimensional component of the polynomial system. Unfortunately, the inverse is not necessarily true, as discussed in [13].

1.4 Related Work

We address related mathematics and software packages for each of the sections of this thesis independently.
1.4.1 Tropical Prevarieties

Computing a tropical prevariety can be considered as a generalization of the mixed volume computation. For the relationship between triangulations and mixed subdivisions, we refer to [24]. In the mixed volume computation, one performs many intersections of closed polyhedral cones. Linear programming models to prune superfluous edge-edge combinations were proposed first in [30]; further developments were made in [35] and [71]. A recent complexity study appeared in [68], along with a report on a parallel implementation of the mixed volume computation.

Related Software. A practical study on various software packages for exact volume computation of a polytope is described in [22]. The authors of [31] present an experimental study of approximate polytope volume computation. In [32], a total polynomial-time algorithm is presented to compute the edge skeleton of a polytope.

Free dedicated software packages to compute mixed volumes are MixedVol [36] and DE-MiCS [70]. In [70], the computation of the mixed volume for the cyclic 16-roots problem was reported for the first time. The mixed volume computation is included in PHCpack [82], pss5 [68], and gfanlib [55] [51]. Gfan [54] contains software to compute the common refinement of the normal fans of Newton polytopes. Gfan relies on cddlib [34] and optionally SoPlex [90] for lower level polyhedral computations.

1.4.2 Monodromy

There exist several software packages designed to perform homotopy continuation, such as PHCpack [82], Bertini [8] and NumericalAlgebraicGeometry for Macaulay2 [62]. In practice,
these software packages find all solutions to a polynomial system faster than a Gröbner basis. In general, polynomial homotopy continuation is a pleasingly parallel task, as all of the paths can be tracked independently (84).

Existing implementations of a numerical irreducible decomposition in PHCpack, Bertini, and NumericalAlgebraicGeometry that use monodromy implement what we will call the naive dynamic strategy. The existing algorithms are in a worse complexity class than what we present in Chapter 4. A parallel implementation of a monodromy algorithm was done in (64).

1.4.3 Computer Algebra in R

R is increasingly the main programming language of the statistics community, but it has limited native support for symbolic computation (74). rSymPy attempts to alleviate this problem by connecting R to Python’s SymPy library (69, 44). mpoly provides a basic collection of R data structures and methods for multivariate polynomials and was designed to lay the foundation for a more robust computer algebra system in R (56).

1.5 Thesis Contribution

This thesis contributes new algorithms and software packages for diverse applications in computational algebraic geometry. Following this introductory chapter, Chapter 2 develops computational polyhedral techniques that will be used for computing tropical prevarieties. Chapter 3 combines the aforementioned methods and develops them into cohesive algorithms. These algorithms have been used to compute several tropical prevarieties that were computationally infeasible prior to this work. This includes the first computation of the tropical prevariety of the cyclic-16 roots problem and the tropical prevariety required for the disproof of the conjecture
by Ren, Shaw, and Sturmfels in (75). Beginning in Chapter 4, we leave tropical computations for more general computations. This chapter develops a new method to find all solutions for a generic polynomial system. Our implementation of this algorithm (28) is competitive with the existing state-of-the-art methods implemented in other software packages. In certain cases, the method performs far better than other algorithms, for example in [18] where our software is the only software that succeeds at computing their examples. Finally, Chapter 5 develops a socket interface between Macaulay2 and R for use by the algebraic statistics community.
CHAPTER 2

A FEW POLYHEDRAL TECHNIQUES

This chapter develops computational techniques from polyhedral geometry while the next chapter integrates them to form several algorithms for computing tropical prevarieties. Chapter 3 concludes with implementation details and experimental results.

2.1 Problem Statement

Given one polynomial in two variables, the Newton-Puiseux algorithm computes series expansions for the algebraic curve defined by the polynomial \((86)\), departing from the edges of the Newton polyhedron. Given a polynomial system, the rays in the tropical prevariety may lead to the series expansions for the positive dimensional solution sets of the system.

Problem Statement: Given a tuple of polynomials \((f_1, f_2, \ldots, f_N)\) of \(n\) variables with normal fans \((F_1, F_2, \ldots, F_N)\), efficiently compute the tropical prevariety with a parallel algorithm.

2.2 An Enumeration Tree

Though not explicit in the definition of the tropical prevariety, computing a tropical prevariety is equivalent to traversing an enumeration tree. To traverse the enumeration tree, it is necessary to define two functions:

1. a function that takes in a cone and determines which is the next tropical hypersurface it should be intersected with, and

2. a function that takes in a cone and a tropical hypersurface and intersects them.
Basic examples of these functions can be seen in Algorithm 1 and in Algorithm 2.

Algorithm 1 Pick the next layer in the enumeration tree

**Input:** A cone $C$ at the $i^{th}$ level of the tree.
**Output:** A $\mathcal{T}(f_j)$ for the $(i+1)^{th}$ level of the tree.

function PickNextTropicalHypersurface
    Result := $\mathcal{T}(f_{i+1})$
end function

Algorithm 2 Perform $C \wedge \mathcal{T}(f_i)$

**Input:** A cone $C$ and a tropical hypersurface $\mathcal{T}(f_i)$.
**Output:** The (possibly empty) set of cones $C \wedge \mathcal{T}(f_i)$.

function IntersectConeWithTropicalHypersurface($C$, $\mathcal{T}(f_i)$)
    Intersect $C$ with each cone of $\mathcal{T}(f_i)$.
    Result := Nonempty cones of $C \wedge \mathcal{T}(f_i)$.
end function

Algorithm 3 shows how these two functions can be called repeatedly to compute a tropical prevariety. Throughout the rest of this chapter, we will work towards developing better methods for both Algorithm 1 and Algorithm 2, though we will not always be explicit about which we are working towards improving.
Algorithm 3 Basic Tropical Prevariety Computation

Input: $F = (f_1, f_2, \ldots, f_N)$
Output: $\mathcal{T}(F)$

function TraverseEnumerationTree($C$)
  if $C$ has visited all $N$ tropical hypersurfaces then
    Output $C$
  end if
  5: Call PickNextTropicalHypersurface to determine which $\mathcal{T}(f_j)$ to visit next.
  for each $C_{new}$ in IntersectConeWithTropicalHypersurface($C$, $\mathcal{T}(f_j)$) do
    TraverseEnumerationTree($C_{new}$)
  end for
end function

10: for each $C \in \mathcal{T}(f_1)$ do
    TraverseEnumerationTree($C$)
end for

2.3 Closed Polyhedral Cones

A closed polyhedral cone in $\mathbb{R}^n$ is a set of the form

$$\{ x \in \mathbb{R}^n \mid Ax \leq 0 \}$$  \hspace{1cm} (2.1)

with $A \in \mathbb{R}^{m \times n}$ and the comparison done coordinatewise. Given a polynomial, it is straightforward to first compute its Newton polytope and then compute its normal fan of closed polyhedral edge cones; the most computationally difficult step in that process is the computation of the convex hull of the set of points. Figure 4 shows an example.

There are some properties of closed polyhedral cones that are useful for the computation of a tropical prevariety (one of which will be discussed in 2.6), but in general it is inconvenient
that all adjacent normal cones have non trivial intersections. This is exacerbated when all fans share a lineality space. In this case, every possible pair of cones intersect, which dramatically slows the computation. In this case, we cannot prune any cones, and we perform \( \prod_{i=1}^{N} |\mathcal{N}(f_i)| \) cone intersections.

Figure 4: *Left:* A Newton polytope \( P \). *Center:* The normal fan \( F \) of \( P \). *Right:* \( F \) split apart into four closed cones. Note that every pair of adjacent cones intersect nontrivially.

2.4 Half Open Polyhedral Cones

In contrast to closed cones, it is also possible to represent the support of a fan as a set of mutually disjoint cones. Such representations were also used in the computation of the tropical prevariety in \([23]\). The key element in the representation is the notion of a half open cone, which is a set of the form
\{ x \in \mathbb{R}^n \mid Ax \leq 0 \land A'x < 0 \} \quad (2.2)

with $A \in \mathbb{R}^{m \times n}$ and $A' \in \mathbb{R}^{m' \times n}$.

### 2.4.1 Constructing Half Open Cones

We provide algorithms that divide the support of fans defined by a convex polytope $P$ into a disjoint set of half open cones. We first explain how this is done for the normal fan of $P$ and later how it is done for the tropical hypersurface of $\mathcal{T}(P)$ of $P$, by which we mean the tropical hypersurface of a polynomial with Newton polytope $P$. Figure 5 shows an example of dividing the support of the normal fan of a single polytope into half open cones.

![Figure 5: Left: A Newton polytope $P$. Center: The normal fan $F$ of $P$. Right: $F$ split apart into a disjoint union of four half open cones. The dashed lines represent boundaries that are not contained by a cone, while the solid lines represent boundaries that are contained by a cone. In this example, the upper right cone contains the origin.](image-url)
We begin by orienting the edge graph of the polytope using a random vector \( r \in \mathbb{R}^n \) and then ordering the vertices by inner product with \( r \). Assuming that these inner products are different, we find a unique sink orientation of the graph by giving each edge of the polytope a direction based on the vertex ordering, as in Figure 6. For a sufficiently random \( r \), this assumption will always hold.

![Figure 6: A cube oriented by a vector \( r \).](image)

Using the unique sink orientation, we define a half open normal cone to each vertex \( v \) of \( P \) in this way: for each edge \( e \) incident to \( v \), create an inequality from it, making it strict if \( e \) is outgoing and non-strict if it is ingoing. The collection of all cones obtained as \( v \) varies divides \( \mathbb{R}^n \) into a disjoint union. For the cones to cover \( \mathbb{R}^n \), it is essential that the oriented graph is cycle free. This is guaranteed by the construction.
To write the tropical hypersurface $T(P)$ as a disjoint union of half open cones, call Algorithm 4 below on each half open cone $C$ constructed above. The output is a collection of half open cones covering exactly the non-interior points of $C$. Taking the union of all these collections as $C$ varies results in a set of cones covering the support of $T(P)$ exactly and containing one cone for each edge of $P$.

---

**Algorithm 4** Create half open cones of codimension one from a full dimensional half open cone.

**Input:** An inequality description of a full dimensional half open cone $C$

**Output:** A collection of disjoint half open cones with union equal to the boundary of $C$

```plaintext
function CREATEHALFOPENCONES($C$)
    if $C$ has only strict constraints then return $\emptyset$
    else
        Choose a non-strict constraint $c$ of $C$
        $C_\prec := C$ but with $c$ being strict
        $C_\equiv := C$ but with $c$ being an equation
        return $C_\equiv \cup$ CREATEHALFOPENCONES($C_\prec$)
    end if
end function
```
2.4.2 **Representation as a Closed Cone**

Linear programming involves sets of equations and non-strict inequalities defining closed polyhedra. Therefore we represent a half open cone $C$ defined by matrices $A$ and $A'$ from (Equation 2.2) by the closed cone $C' \subseteq \mathbb{R}^{n+1}$ defined by the matrix

$$
\begin{bmatrix}
A & 0 \\
A' & 1
\end{bmatrix}
$$

with a column of zeros and ones appended. Then $C = \pi(C' \cap (\mathbb{R}^n \times \mathbb{R}_{>0}))$ where $\pi : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^n$ denotes the projection forgetting the last coordinate. Observing that for all $x \in C$ we have $x \times \varepsilon \in C'$ for $\varepsilon > 0$ sufficiently small, we obtain $\overline{C} \subseteq \pi(C' \cap (\mathbb{R}^n \times \{0\}))$, while the converse inclusion holds if and only if $C$ is non-empty. This happens if and only if $C' \cap (\mathbb{R}^n \times \mathbb{R}_{>0}) \neq \emptyset$ and in that case $\dim(C') = \dim(C) + 1$. Non-emptiness and other properties can be determined with linear programming.

2.5 **Relation Tables**

In the algorithms we present in Chapter 3, we will use relation tables in two distinct ways. At this juncture, we will merely define what we mean by them and how to construct them, not justify their usefulness (however, we note that they can be used to improve both Algorithm 1 and Algorithm 2). Relation tables were initially introduced in the mixed volume literature in [35]. In MixedVol, relation tables were initialized at the beginning of the mixed volume computation and remained static throughout the remainder of the computation.
Let $C_{i,j}$ be the $j^{th}$ cone of the $i^{th}$ tropical hypersurface. A relation table is a chunk of computer memory that contains information about whether or not a single cone $C_{i,j}$ can intersect with each other cone. For each $(k,l)$, $1 \leq k \leq \#$Tropical hypersurfaces and $1 \leq l \leq |T(P_i)|$, write Boolean values into this chunk of memory by the following rule:

$$0, \text{ if } C_{i,j} \cap C_{k,l} = \emptyset$$

$$1, \text{ if } C_{i,j} \cap C_{k,l} \neq \emptyset$$

$$0, \text{ if } i = k.$$  

We choose to compute the relation table for each cone of each fan at the beginning of some of the algorithms in Chapter 3 Algorithm 5 formalizes this process. Note that it is faster to check if $C_{i,j} \cap C_{k,l} = \emptyset$ than it is to compute $C_{i,j} \cap C_{k,l}$ with redundant inequalities removed. Checking if an intersection is empty is equivalent to checking the feasibility of a linear system, while computing the intersection of two cones is equivalent to computing a convex hull.

When we intersect cone objects in some of the algorithms of Chapter 3, we will intersect the polyhedral cones and we will also intersect the associated relation tables. Intersecting relation tables requires creating a new bit array that is equal to a bitwise AND of the two input bit arrays, as shown in Figure 7. As this is a bit operation, it is very fast. If the new relation table has a 0 in some position, then it is certain that the new cone cannot intersect the cone associated to that position. However, if the new relation table has a 1 at a position, it is possible that the new cone could not intersect the cone associated to that position. For example, suppose that we have three fans, each with one cone as in Figure 8.
Algorithm 5 Create a relation table for each cone in each of the given fans.

Input: \( \mathcal{T}(P_1), \ldots, \mathcal{T}(P_n) \)

procedure CREATE_RELATION_TABLES
    for \( i \) from 1 to \( n \) do
        for \( j \) from 1 to \( |\mathcal{T}(P_i)| \) do
            Instantiate the relation table of \( C_{i,j} \) with \( (\sum_{k=1}^{n} |\mathcal{T}(P_k)|) \) 0’s.
        end for
    end for
    for \( i \) from 1 to \( n - 1 \) do
        for \( j \) from 1 to \( |\mathcal{T}(P_i)| \) do
            for \( k \) from \( i + 1 \) to \( n \) do
                for \( l \) from 1 to \( |\mathcal{T}(P_k)| \) do
                    if \( C_{i,j} \cap C_{k,l} \neq \emptyset \) then
                        Fill in 1 in the appropriate location in both relation tables.
                    end if
                end for
            end for
        end for
    end for
end procedure

\begin{center}
\begin{tabular}{llllllll}
& 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
\hline
\hline
C relation table & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
\hline
C_{i,j} relation table & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
\hline
C \cap C_{i,j} relation table & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{tabular}
\end{center}

Figure 7: Sample intersection of two relation tables. Each relation table is an array of bits, so intersecting a pair of relation tables is a bitwise AND.
Figure 8: Cones and their associated relation tables. Each cone is the sole cone from a distinct fan.

Suppose $C_{1,1}$ is intersected with $C_{2,1}$. The resulting cone has the inequalities and relation table written in Figure 9. However, this new cone has a trivial intersection with $C_{3,1}$, even though the relation table does not indicate that with a 0. Because of this shortcoming, the relation tables are more of guidelines than actual rules.

$$ C_{1,1} = \begin{cases} x \geq 0 \\ y \geq 0 \\ z = 0 \end{cases} \quad C_{2,1} = \begin{cases} x \geq 0 \\ y = 0 \\ z \geq 0 \end{cases} \quad C_{3,1} = \begin{cases} x = 0 \\ y \geq 0 \\ z \geq 0 \end{cases} $$

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 9: $C_{1,1} \cap C_{2,1}$.
2.6 **Traversing the Edge Skeleton of a Newton Polytope**

We now describe a new polyhedral method that leads to a more efficient way to intersect a cone with a tropical hypersurface, as required by Algorithm 2. To explain this method, we will define a new term, prove a theorem about it, give an example, and finish with a formally correct algorithm. Let a *pretropism graph* be the set of edges for a Newton polytope that have closed normal cones intersecting some cone $C$.

**Theorem 2.6.1.** Pretropism graphs are connected graphs.

*Proof.* Let $C$ be a cone, and let $P$ be a polytope with edges $e_1, e_2$ such that they are in the pretropism graph of $C$. Let $C_1$ be the cone of the intersection of the normal cone of $e_1$ with $C$, and let $C_2$ be the cone of the intersection of the normal cone of $e_2$ and $C$. If we can show that there exists a path between $e_1$ and $e_2$ that remains in the pretropism graph, then the result will follow.

Let $n_1$ be a normal to $e_1$ that is also in $C_1$ and let $n_2$ be a normal to $e_2$ that is also in $C_2$. Set $n = tn_1 + (1-t)n_2$ where $0 \leq t \leq 1$. Consider varying $t$ from 0 to 1; this creates the cone $C_n$, a cone which must lie within $C$, as both $n_1$ and $n_2$ lie in that cone. As $n$ moves from 0 to 1, it will progressively intersect new faces of $P$ that have all of their edges in the pretropism graph. Eventually, this process terminates when we reach $e_2$, and we have constructed a path from $e_1$ to $e_2$. Since a path always exists, the result follows. \(\square\)

Since pretropism graphs are connected graphs, to intersect $C$ with $\mathcal{T}(f)$ we simply need to find part of the pretropism graph and then exploit the connectivity of the edge skeleton to find the rest of it. In Figure 10 we return to our example from Chapter 1. Suppose $C$ is the cone
from the square pyramid (a), and we would like to intersect it with the tropical hypersurface defined by the cube (b). We begin by picking a random ray $r$ in the interior of $C$ and taking an initial form of the cube with respect to that ray. Suppose that the initial form is a single vertex, as in (c). We then consider all of the edges that are connected to that vertex, edges 1, 4, and 8 as in (d). If we intersect each of those three edge cones with $C$, we see that only the edge cone for edge 4 has a nontrivial intersection with $C$ (e). We next visit all of the edges connected to 4 that we have not yet considered, which are edges 3 and 7 as in (f). We see that neither of those two edge cones intersect with $C$, so we terminate. We have performed five cone intersections to intersect $C$ with the tropical hypersurface defined by the cube. If we had done this intersection naively, we would intersect $C$ with each of the twelve codimension one cones from the cube.

Algorithm 6 formalizes the process we have introduced. The exploration of the neighboring edges corresponds to tilting the ray $r$ in Algorithm 6 as in rotating a hyperplane in the gift wrapping method (81). One may wonder why the exploration of the edge skeleton in Algorithm 6 needs to continue after the statement on line 4. This is because the cone $C$ has the potential to intersect many cones in $P$, particularly if $P$ has small cones or $C$ is large.

Through a result by Jensen (51) which generalizes an earlier result by Osserman and Payne (73), we can make a small change to Algorithm 6 which leads to large computational speedup. Specifically, line 11 of Algorithm 6 can be replaced with

$$\text{if } \dim(C \cap C_E) - 1 \geq \min\{\dim(C), \dim(C_E)\}.$$
Algorithm 6 Exploit the connectivity of the edge skeleton of a Newton polytope to compute $C \cap T(f_i)$.

**Input:** A cone $C$ and a tropical hypersurface $T(f_i)$.

**Output:** The (possibly empty) set of cones $C \cap T(f_i)$.

```plaintext
function EXPLOREEDGESkeleton($C$, $T(f_i)$)
    $P := NP(f)$
    $r := $ a random ray inside $C$
    $in_r(P) :=$ vertices of $P$ that have minimal inner product with $r$
    EdgesToTest := all edges of $P$ that have vertices in $in_r(P)$
    Cones := $\emptyset$
    TestedEdges := $\emptyset$
    while EdgesToTest $\neq \emptyset$
        $E :=$ pop an edge from EdgesToTest
        $C_E :=$ normal cone to $E$
        if $C \cap C_E \neq \{0\}$ then
            Cones := Cones $\cup$ ($C \cap C_E$)
            Edges := Edges $\cup$ $E$
            for each neighboring edge $e$ of $E$ do
                if $e \notin$ TestedEdges then
                    EdgesToTest := EdgesToTest $\cup$ $e$
                end if
            end for
        end if
        TestedEdges := TestedEdges $\cup$ $E$
    end while
    return Cones
end function
```

This result applies more broadly, anywhere we are intersecting a cone with a polyhedral fan, where the polyhedral fan consists of closed cones. Unfortunately, this result does not apply to half open cones.
Figure 10: Example of traversing the edge skeleton.
CHAPTER 3

ALGORITHMS FOR COMPUTING

TROPICAL PREVARIETIES

We will now define three algorithms for computing tropical prevarieties. The first is not practical as it becomes computationally intensive prohibitively quickly. The latter two algorithms are both useful in practice; notably, the usefulness of Algorithm 9 will be demonstrated in Section 3.4 through the first computation of the tropical prevariety of cyclic-16 and through the disproof of the conjecture of Ren, Shaw, and Sturmfels in (75).

3.1 Algorithm 1: The Combinatorial Algorithm

The common refinement in the definition of the tropical prevariety has a constructive formulation: compute the intersection of every combination of \( N \) cones, one from each fan, and add the non-empty intersections to the output. This basic algorithm requires \( \prod_{i=1}^{N} |T(f_i)| \) cone intersections. The recursive formulation of a combinatorial algorithm that we present in Algorithm 7 (which we call static enumeration) performs substantially fewer cone intersections than \( \prod_{i=1}^{N} |T(f_i)| \).

The recursive execution of Algorithm 7 leads to a tree of cone intersections with a de facto depth first traversal. At the \( i \)th level of the tree are a set of cones comprising the common refinement of the first \( i \) fans. If the cones in Algorithm 7 are closed, then the intersections are
Algorithm 7 Static enumeration

**Input:** A list \( F \) of fans \( F_1, \ldots, F_N \) in \( \mathbb{R}^n \) where each \( F_i \) is represented by a list of cones covering the support of \( F_i \).

**Output:** A list of cones covering the support of \( F_1 \land \cdots \land F_N \).

**procedure** `STATICENUMERATION(Cone C, Index i)`

1. if \( C \neq \emptyset \) then
   1.1. if \( i > |F| \) then
       1.1.1. Output \( C \)
   1.2. else
       1.2.1. for each cone \( D \) in \( F_i \) do
           1.2.2. `STATICENUMERATION(C \cap D, i + 1)`
       1.2.3. end for
   1.3. end if
1.4. end if

**end procedure**

`STATICENUMERATION(\mathbb{R}^n, 1)`

always non-empty, though possibly trivial. One may wish to remove duplicate cones at each level of the tree.

If the fans \( F_i \) in Algorithm 7 each consist of mutually disjoint half open cones, then the occurrence of duplicate intersections is avoided and cone intersections can be empty. Thus, working with half open cones in Algorithm 7 brings a substantial improvement.

### 3.2 Algorithm 2: Traversing the Newton Polytopes

Algorithm 8 sketches the outline of our second algorithm to compute the tropical prevariety of a set of \( N \) polytopes. Along the lines of the gift wrapping algorithm, for every edge of the first polytope we take the plane that contains this edge and consider where this plane touches the second polytope. The core helper function that gets called repeatedly is Algorithm 6, which explores the edge skeleton of a polytope.
Algorithm 8

**Input:** A list of Newton polytopes $P_1, \ldots, P_N$ in $\mathbb{R}^n$.

**Output:** A list of cones covering the tropical prevariety.

**function** TROPICAL-PREVARIETY($P_1, \ldots, P_N$)

Cones := codimension one normal cones of $P_1$

for $i := 2$ to $N$ do

NewCones := $\emptyset$

for Cone in Cones do

NewCones := NewCones $\cup$ EXPLORE-EDGE-SKELETON(Cone, $T(P_i)$)

end for

for Cone in NewCones do

if Cone is contained within another cone in NewCones then

NewCones := NewCones - Cone

end if

end for

Cones := NewCones

end for

return Cones

end function

3.2.1 Duplicate Cones

Algorithm 8 does not work with half open cones, so it is necessary for an efficient algorithm to remove duplicate cones at each level of the tree traversal. Lines 8 through 12 in the algorithm achieve this goal. They implement a basic method, which is the $O(|\text{NewCones}|^2)$ method of checking every pair of cones and marking redundant cones for removal. More sophisticated methods are possible, including finding canonical representations of cones and using search tries. However, as the size of the tropical prevariety increases, the time and space required to perform this part of the algorithm becomes large.
3.2.2 Analysis of Computational Complexity

To estimate the cost of Algorithm 8, we will first consider the case when there are two polytopes. We will take the primitive operation of computing the tropical prevariety to be the number of cone intersections performed, as that number will drive the run time of the algorithm. The upper bound on the number of primitive operations for two polytopes \( P_1 \) and \( P_2 \) is the product \(|P_1| \times |P_2|\), while the lower bound equals the number of maximal cones in the tropical prevariety.

Denote by \( G(P, C) \) the pretropism graph resting on polytope \( P \) defined by some input cone \( C \). Define \(|G(P, C)|\) to be the number of edges in \( G(P, C) \).

**Proposition 3.2.1.** The number of primitive operations in Algorithm 8 on two polytopes \( P_1 \) and \( P_2 \) is approximately

\[
\sum_{i=1}^{|P_1|} |G(P_2, C_i)|,
\]

where \( C_i \) is the \( i^{th} \) codimension one normal cone of \( P_1 \).

Since \( G(P, C) \) is a subset of the edges of \( P \), \(|G(P, C)| \leq |P|\). The number of primitive operations in (Equation 3.1) is approximate because to find the boundary of a pretropism graph, it is necessary to find the perimeter as in Algorithm 6.

To interpret Equation 3.1 recall that Algorithm 8 takes a ray from inside a normal cone to an edge of the first polytope for the exploration of the edge graph of the second polytope. If we take a simplified view on the second polytopes as a ball, then shining a light on that
ball will illuminate at most half of its surface. If we use the estimate: \(|G_{P_2, C_1}| \approx |P_2|/2\), then Algorithm \[^8\] cuts the upper bound on the number of primitive operations in half.

Estimating the cost of the \(N\) polytope case follows naturally from the cost analysis of the two polytope case. For \(N\) polytopes, the upper bound on the number of primitive operations required is the product \(|P_1| \times |P_2| \times \ldots \times |P_N|\).

**Proposition 3.2.2.** The number of primitive operations in Algorithm \[^8\] on \(N\) polytopes \(P_1, P_2, \ldots, P_N\) is approximately

\[
\sum_{i=1}^{N} \left( \prod_{j=2}^{N} \left| G_{P_j, C_i} \right| \right)
\]

(3.2)

where \(C_i\) is the \(i^{th}\) codimension one cone of \(P_1\).

Again, if we use the estimate that \(|G_{P_j, C_i}| \approx |P_j|/2\), then Algorithm \[^8\] reduces the expected number of primitive operations by \(\frac{1}{2^{N-1}}\). This estimate depends entirely on the intuition that we are cutting the number of comparisons in half. In practice, we found this predicted value to be too conservative. We wrote an implementation of this algorithm so we could test the aforementioned estimate; the C++ code is available at [https://github.com/sommars/TropicalPrevariety](https://github.com/sommars/TropicalPrevariety). For various \(N\), we created \(N - 1\) generic simplices of dimension \(N\) spanned by integer points with coordinates uniformly generated within the range 0 to 30. When we ran Algorithm \[^8\] we found that it scaled even better than predicted, as shown in Table I.
3.3 Algorithm 3: Dynamic Enumeration

Our final algorithm for computing tropical prevarieties draws inspiration from the mixed volume literature to reduce the number of cone intersections through the use of dynamic enumeration (35; 71). Dynamic enumeration can be viewed as a greedy method to reorder the fans during the computation, thereby affecting the shape of our enumeration tree (i.e. modifying Algorithm 1). A random permutation of the \( N \) polytopes before the start of Algorithm 7 defines a sub-optimal order in which to intersect cones. Using a greedy metric, defined in Section 3.3.1, we determine with which fan it is best to start.

Every intermediate cone intersection \( C \) must be intersected with each remaining fan in order to contribute to the final common refinement, but it can be intersected with the fans in any order. We greedily choose the next fan to intersect with \( C \), as a fan with which \( C \) is expected to have few non-empty intersections. After intersecting with the fan we get a new set of cones, each contained within \( C \). Each of these new cones must be intersected with the remaining

<table>
<thead>
<tr>
<th>( N )</th>
<th>( \frac{1}{2^{N-1}} )</th>
<th>Actual Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.72</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>0.288</td>
</tr>
<tr>
<td>5</td>
<td>0.125</td>
<td>0.0478</td>
</tr>
<tr>
<td>6</td>
<td>0.0625</td>
<td>0.00452</td>
</tr>
<tr>
<td>7</td>
<td>0.03125</td>
<td>0.000301</td>
</tr>
<tr>
<td>8</td>
<td>0.015625</td>
<td>0.0000147</td>
</tr>
</tbody>
</table>

TABLE I: For \( N - 1 \) generic simplices, the predicted and actual ratio of number cone intersections of the definitional algorithm to the number required by Algorithm 8.
Algorithm 9 Dynamic enumeration sketch

Input: A list $F$ of fans $F_1, \ldots, F_N$ in $\mathbb{R}^n$ where each $F_i$ is represented by a list of cones covering the support of $F_i$.

Output: A list of cones covering the support of $F_1 \land \cdots \land F_N$.

procedure DYNAMIC_ENUMERATION(Cone $C$, Set $I$)
    if $C \neq \emptyset$ then
        if $I = \emptyset$ then
            Output $C$
        else
            Greedily choose index $i \in I$.
            for each cone $D$ in $F_i$ do
                DYNAMIC_ENUMERATION($C \cap D$, $I \setminus \{i\}$)
            end for
        end if
    end if
end procedure

DYNAMIC_ENUMERATION($\mathbb{R}^n$, $\{1, \ldots, |F|\}$)

fans, but this can happen in whatever order seems to be the most efficient, determined through greedy selection. This process ends when $C$ is the result of an intersection of a cone from each fan.

Algorithm 7 and Algorithm 9 have recursion trees with different shapes. In the setting of mixed volume computation it was observed in (70) that the tree of Algorithm 9 has far fewer vertices. Figure 11 demonstrates the difference between the tree traversal of static enumeration and dynamic enumeration.

3.3.1 Greedy Selection

The basic unit of work for Algorithm 7 and Algorithm 9 is intersecting a pair of polyhedral cones. Every greedy choice we make is done to minimize the number of necessary intersections
Figure 11: These trees illustrate the difference between static and dynamic enumeration for three fans $F_1$, $F_2$, and $F_3$. The left tree represents static enumeration, where the ordering of the fans is established and does not change. The right tree represents dynamic enumeration, where the starting fan is greedily selected, and each cone greedily chooses which fan to be intersected with next.

while avoiding adding an additional computational burden. To this end, the choice of the first fan is easy: pick the fan with the fewest cones.

The greedy metric used during the tree traversal is more difficult. To make the greedy choices that our algorithm requires, our software consults a cone’s relation table to find the unused fan with the fewest cones with which it could intersect. This greedy selection requires minimal additional computation and leads to large speedups as will be shown in Section 3.4.2. An additional benefit of the relation tables is that they allow us to avoid intersecting cones that are already known to not intersect. This is illustrated in Algorithm [10].

Note that dynamic enumeration works best with fans made up of disjoint sets of half open cones. If the fans consist of closed cones, duplicate cones will be found, but it is impossible
Algorithm 10 Full dynamic enumeration algorithm

**Input:** A list \( F \) of fans \( F_1, \ldots, F_N \) in \( \mathbb{R}^n \) where each \( F_i \) is represented by a list of cones covering the support of \( F_i \).

**Output:** A list of cones covering the support of \( F_1 \wedge \cdots \wedge F_N \).

```
procedure DynamicEnumeration(Cone \( C \), Set \( I \))
    if \( C \neq \emptyset \) then
        if \( I = \emptyset \) then
            Output \( C \)
        else
            Choose index \( i \in I \) such that \( F_i \) has fewest cones which \( C \) could intersect.
            for each cone \( D \) in \( F_i \) do
                if \( C \)'s relation table allows \( C \cap D \neq \emptyset \) then
                    Intersect \( C \)'s relation table with \( D \)'s relation table, and store on \( C \cap D \)
                    DynamicEnumeration(\( C \cap D \), \( I \setminus \{i\} \))
                end if
            end for
        end if
    end if
end procedure
```

Compute relation tables for \( \mathbb{R}^n \) and the cones in \( F \)

```
DynamicEnumeration(\( \mathbb{R}^n \), \{1, \ldots, |F|\})
```

to detect and remove them mid-algorithm like in Section 3.2. Because of this, the required computational time and space will bloat. Unfortunately, this means that it is not practical to combine dynamic enumeration and traversing the edge skeleton of a Newton polytope, as they require different types of polyhedral cones.
3.4 DynamicPrevariety

We created the software package DynamicPrevariety to implement a parallel version of Algorithm 10. The source code is available at https://github.com/sommars/DynamicPrevariety, and it is licensed under the GNU General Public License (GPL) version 3.

DynamicPrevariety uses the Parma Polyhedra Library (PPL) (7) for its linear programming library, and in particular its thread safe multithreading capabilities. Speedups were obtained through using the thread-caching malloc TCMalloc (39), particularly when PPL was used in parallel.

3.4.1 Parallel Implementation

While the polytopes, fans, and cones in the tropical previariety all live in $\mathbb{R}^n$, we point out that their defining data is exact, spanned by points with integer coordinates. All our computations are performed with arbitrary precision integer arithmetic. The rapid coefficient growth is not polynomial in the dimension, but the dimension is relatively small in our applications.

Our choices of software packages both enables and limits our options for a parallel implementation. We chose to use PPL for all polyhedral computations (7). We first call it to find the vertices of the Newton polytopes from the support sets, and we later call it many times to compute intersections of polyhedral cones. PPL uses arbitrary precision integers during its computations via the GNU Multi Precision Arithmetic Library (GMP) (40). If we had chosen to use a library that did not use arbitrary precision integers, the software would need to exercise care and certify that there were no incorrect answers due to floating point error. Since computing a tropical previariety requires intersecting many polyhedral cones in sequence, as the depth
of the tree increases, the likelihood of floating point error also increases. Using a polyhedral library that uses GMP integers avoids this complication.

PPL has recently become threadsafe, which allows us to use it in a parallel implementation. Unfortunately, if multiple threads are intersecting pairs of polyhedra simultaneously, linear speedups are not be achieved. This is due to the fact that when multiple threads attempt to allocate GMP integers from the heap, a linear speedup is not attained. Modest improvements in speedup come from using the allocator TCMalloc (39), which reduces the time required for each allocation.

For computing a tropical prevariety, we distinguish three stages in the algorithm. For each of the three stages, we consider its parallel execution.

1. The algorithm begins by computing the vertices of the Newton polytopes, which is necessary to determine the normal fans. Computing the vertices of distinct polytopes can be done in parallel, but we have found it not to be necessary. The polynomial systems of interest are sparse, with small Newton polytopes spanned by relatively few monomials; if this were not the case, determining the vertices would be more difficult. In our most computationally intensive benchmark, computing the vertices takes less than a second for a single thread. Therefore, this component of the algorithm has not been implemented in parallel.

2. The second stage we consider is the computation of the relation tables. Filling the relation tables requires computing the intersection of many pairs of polyhedral cones and testing if that intersection is non-empty. This can be viewed as a job queue, where each job is
the intersection of two polyhedral cones. The queue is filled with all of the necessary jobs, then each process pops a job, records the result of the intersection and returns to the queue. This process continues until the job queue has been emptied.

3. The third stage of the algorithm leads to the greatest benefit: developing a parallel version of the recursion in Algorithm 10. This will be addressed in the remainder of this section.

A first, coarse grained parallel version of Algorithm 10 was implemented using forked processes, dividing the cones of the starting fan among several processes. Each process took its starting cones and performed Algorithm 10 on them, terminating when finished. This approach was a natural starting point, as it did not require communication between threads and it was straightforward to implement. Since the processes were distinct, each thread had its own heap, so we were closer to achieving linear speedups in the polyhedral computations. However, the time required for each process varied dramatically. For a run of the cyclic-16 roots polynomial system with twenty threads, the fastest threads finished in less than a day while the slowest thread took more than three weeks to finish. This was an inefficient use of resources, as a good parallel implementation uses all of a computer’s available resources for the duration of the computation.

Our current parallel implementation applies work stealing [15], using the run time parallel library provided by PPL. The first barrier to creating a work stealing implementation of the dynamic enumeration method is that Algorithm 10 is a recursive algorithm, so it lacks a job queue. We define a single job to be taking a cone and intersecting it with the normal cones of a polytope (i.e. a job is an implementation of Algorithm 2); Algorithm 11 transforms
Algorithm into an algorithm with a work queue of these jobs. This version of Algorithm will find the prevariety with the same number of cone intersections, but it will find the cones of the prevariety in a different order.

\textbf{Algorithm 11} Iterative version of dynamic enumeration

\textbf{Input:} A list of fans $F_1, \ldots, F_N$ in $\mathbb{R}^n$ where each $F_i$ is represented by a list of cones covering the support of $F_i$.

\textbf{Output:} A list of cones covering the support of $F_1 \land \cdots \land F_N$.

Compute relation tables

$F :=$ fan with fewest cones

Cones := Cones from $F$

\textbf{while} Cones $\neq \emptyset$ \textbf{do}

5: \quad $C :=$ remove an element from Cones

Choose fan $F'$ not used to produce $C$ such that $F'$ has fewest cones with which $C$ could intersect.

\textbf{for} each cone $D$ in $F'$ \textbf{do}

\hspace{1em} \textbf{if} $C$’s relation table allows $C \cap D \neq \emptyset$ \textbf{then}

\hspace{2em} Compute $C \cap D$

\hspace{2em} \textbf{if} $C \cap D \neq \emptyset$ \textbf{then}

\hspace{3em} \textbf{if} $C \cap D$ used all fans \textbf{then}

\hspace{4em} Output $C \cap D$

\hspace{3em} \textbf{else}

\hspace{4em} Intersect $C$’s relation table with $D$’s relation table, and store on $C \cap D$

\hspace{4em} Add $C \cap D$ to Cones

\hspace{2em} \textbf{end if}

\hspace{2em} \textbf{end if}

\hspace{1em} \textbf{end for}

10: \quad \textbf{end if}

20: \quad \textbf{end while}
There are two benefits to finding cones in the tropical prevariety quickly. Once a cone in the prevariety has been discovered, it can be written to file, so the memory that it consumed can be freed. Additionally, when cones in the prevariety are found, post-processing can begin, which may vary depending on the application. One application of interest is computing power series expansions of positive dimensional solution sets of polynomial systems. Each cone could lead to several distinct power series, thus this process could begin as soon as a single cone has been found.

To find cones in the prevariety as quickly as possible, the job queue is implemented as in Figure 12, thereby also making it more stack-like. When Algorithm 11 begins, cones are placed into an initial subqueue, subqueue 1. When a cone from subqueue 1 is removed and has been intersected with a set of cones from another polytope, the resulting cones are put into subqueue 2. To achieve the goal of finding cones in the prevariety quickly, when choosing the next job, an optimal strategy requires picking a cone from the subqueue with the highest index.

To transform this algorithm into a multi-threaded work stealing algorithm, each thread must have its own queue in the style of Figure 12. When picking a job to execute, a thread first looks to its own queue and picks a cone from the subqueue of highest index. When a thread’s queue is empty, it looks to steal from another thread’s queue, but steals the job from the subqueue of lowest index, as to require stealing less often. Furthermore, if there are \( j \) total threads, the \( i \)th thread looks to steal from threads in the following order: \( i + 1, i + 2, \ldots, j, 1, 2, \ldots i - 1 \). This avoids having all threads attempting to steal from the same thread, keeping the theft spread out among different threads and requiring fewer total robberies.
Figure 12: A queue made up of subqueues. Subqueue 1 contains cones from the starting polytope, while subqueue $N - 1$ contains cones that are the intersection of $N - 1$ cones.

Since there is no communication between the branches of the enumeration tree in Algorithm 10, an alternative to work stealing is to phrase the recursive part of the algorithm as an abstract tree traversal. Then, a general purpose parallel tree traverser can be applied to the problem, which separates the parallelization from the problem domain. This approach was used in (55) for the related problem of mixed volume computation.
3.4.2 Experimental Results

For finding isolated solutions of polynomial systems, there exist many standard benchmark problems. However, few of these problems have positive dimensional components as well, which makes them inappropriate test cases for computing tropical prevarieties. We will mention experimental results from three standard benchmark problems with positive dimensional solution components as well as two challenging problems from tropical geometry.

With the exception of the Gfan timings, all computations were done on a 2.2 GHz Intel Xeon E5-2699 processor in a CentOS Linux workstation with 256 GB RAM using varying numbers of threads.

3.4.2.1 n-body and n-vortex Problems

For equal masses, the central configurations in the classical n-body problem are solutions to the \( \binom{n}{2} \) Albouy-Chenciner equations obtained by clearing denominators of the equations

\[
\sum_{k=1}^{n} (x_{ik}^{-3} - 1)(x_{jk}^2 - x_{ik}^2 - x_{ij}^2) + (x_{jk}^{-3} - 1)(x_{ik}^2 - x_{jk}^2 - x_{ij}^2) = 0
\]

indexed by \( i \) and \( j \) where \( 1 \leq i < j \leq n \), and there are \( \binom{n}{2} \) pairwise distance variables \( x_{12} \ldots x_{(n-1)n} \) and \( x_{ij} = x_{ji} \).
The $n$-vortex problem \cite{47} arose from a generalization of a problem from fluid dynamics that attempted to model vortex filaments. In this setting, the \( \binom{n}{2} \) Albouy-Chenciner equations are obtained by clearing denominators of

\[
\sum_{k=1}^{n} (x_{ik}^{-2} - 1)(x_{jk}^2 - x_{ik}^2 - x_{ij}^2) + (x_{jk}^{-2} - 1)(x_{ik}^2 - x_{jk}^2 - x_{ij}^2) = 0.
\]

Computing a tropical prevariety was essential in the argument of finiteness of the relative equilibria in the 4-body problem \cite{46}. Table II and Table III contain data from experiments with our implementation run on the $n$-body and $n$-vortex equations above. Since the problems increase in difficulty quickly, we can compute only a few tropical prevarieties in each family.

<table>
<thead>
<tr>
<th>$n$</th>
<th>#Rays</th>
<th>1 thread</th>
<th>20 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>0.014s</td>
<td>0.038s</td>
</tr>
<tr>
<td>4</td>
<td>57</td>
<td>0.77s</td>
<td>0.61s</td>
</tr>
<tr>
<td>5</td>
<td>2908</td>
<td>2m37s</td>
<td>34s</td>
</tr>
</tbody>
</table>

TABLE II: $n$-body problem: number of generating rays and timings of DynamicPrevariety run with 1 or 20 threads. The 6-body problem did not terminate in two days when run with 20 threads.
<table>
<thead>
<tr>
<th>n</th>
<th>#Rays</th>
<th>1 thread</th>
<th>20 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>0.011s</td>
<td>0.03s</td>
</tr>
<tr>
<td>4</td>
<td>27</td>
<td>0.44s</td>
<td>0.42s</td>
</tr>
<tr>
<td>5</td>
<td>643</td>
<td>30.1s</td>
<td>10.9s</td>
</tr>
<tr>
<td>6</td>
<td>152,514</td>
<td>3h16m13s</td>
<td></td>
</tr>
</tbody>
</table>

TABLE III: $n$-vortex problem: number of generating rays and timings of DynamicPrevariety run with 1 or 20 threads. The 7-vortex problem did not terminate in two days when run with 20 threads.

3.4.2.2 Cyclic-$n$ Roots

The cyclic-$n$ roots problem asks for the solutions of a polynomial system, commonly formulated as

$$
\begin{cases}
    x_0 + x_1 + \cdots + x_{n-1} = 0 \\
    i = 2, 3, \ldots, n - 1 : \sum_{j=0}^{n-1} \prod_{k=j}^{j+i-1} x_k \mod n = 0 \\
    x_0x_1x_2 \cdots x_{n-1} - 1 = 0.
\end{cases}
$$

(3.3)

This problem is important in the study of biunimodular vectors, a notion that traces back to Gauss, as stated in [33]. In [6], Backelin showed that if $n$ has a divisor that is a square, i.e. if $d^2$ divides $n$ for $d \geq 2$, then there are infinitely many cyclic-$n$ roots. The conjecture of Björck and Saffari [10], [33] Conjecture 1.1) is that if $n$ is not divisible by a square, then the set of cyclic-$n$ roots is finite. If the dimension is a prime number, then the number of solutions is finite, as proven in [45] with an explicit count of the number of solutions given.
The cyclic-\(n\) roots problem scales slowly, so it is a good case study to examine the effectiveness of Algorithm \[10\] in detail. The first two columns of Table IV show that as the size of the problem increases, the benefit of dynamic enumeration increases as well.

<table>
<thead>
<tr>
<th>(n)</th>
<th>Static Enum</th>
<th>Dyn. Enum.</th>
<th>#Rays</th>
<th>Gfan 1 thread</th>
<th>10 threads</th>
<th>20 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>114</td>
<td>114</td>
<td>2</td>
<td>0.020s</td>
<td>0.008s</td>
<td>0.017s</td>
</tr>
<tr>
<td>5</td>
<td>682</td>
<td>676</td>
<td>0</td>
<td>0.058s</td>
<td>0.036s</td>
<td>0.053s</td>
</tr>
<tr>
<td>6</td>
<td>2,286</td>
<td>2,254</td>
<td>8</td>
<td>0.22s</td>
<td>0.10s</td>
<td>0.11s</td>
</tr>
<tr>
<td>7</td>
<td>7,397</td>
<td>7,163</td>
<td>28</td>
<td>0.64s</td>
<td>0.29s</td>
<td>0.26s</td>
</tr>
<tr>
<td>8</td>
<td>19,619</td>
<td>18,315</td>
<td>94</td>
<td>2.87s</td>
<td>0.79s</td>
<td>0.49s</td>
</tr>
<tr>
<td>9</td>
<td>63,109</td>
<td>50,584</td>
<td>276</td>
<td>13.0s</td>
<td>2.8s</td>
<td>1.2s</td>
</tr>
<tr>
<td>10</td>
<td>269,223</td>
<td>160,203</td>
<td>712</td>
<td>1m22s</td>
<td>9.8s</td>
<td>4.4s</td>
</tr>
<tr>
<td>11</td>
<td>1,625,520</td>
<td>827,469</td>
<td>2,244</td>
<td>9m17s</td>
<td>50s</td>
<td>16.8s</td>
</tr>
<tr>
<td>12</td>
<td>11,040,912</td>
<td>5,044,441</td>
<td>5,582</td>
<td>82m33s</td>
<td>5m2s</td>
<td>1m5s</td>
</tr>
<tr>
<td>13</td>
<td>36,633,391</td>
<td>14,872</td>
<td>46m59s</td>
<td>8m30s</td>
<td>6m20s</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>264,463,730</td>
<td>49,114</td>
<td>6h22m56s</td>
<td>67m31s</td>
<td>46m37s</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1,852,158,881</td>
<td>145,276</td>
<td>10h25m45s</td>
<td>7h43m57s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>13,715,434,028</td>
<td>527,126</td>
<td>84h20m37s</td>
<td>62h36m31s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE IV:** This table contains results from experiments with the cyclic-\(n\) roots problem. The left half contains the number of cone intersections required in static enumeration and dynamic enumeration, as well as the number of generating rays i.e. 1-dimensional cones in the produced fan. The right half of the table contains timings of Gfan and the dynamic prevariety software run with 1, 10, or 20 threads. The Gfan timings are for a single thread running on an Intel Xeon E2670 CPU. SoPlex was enabled in the Gfan timings, providing a speed up of roughly a factor 3.

From the right hand portion of Table IV it can be seen that there is a speedup as the number of threads increases. In cyclic-14, a speedup of 5.67 was achieved with ten threads.
while a speedup of 8.21 was achieved with twenty threads. These speedups are not linear, due to the GMP integer allocation issue. However, computing in parallel dramatically reduces computation time, so it is beneficial in practice.

For cyclic-16, there are many cones of high dimension, which makes the prevariety more challenging to compute, see Table V. DynamicPrevariety is the first and so far the only software that can compute the tropical prevariety of the cyclic-16 roots problem. We announced this result in [53].

<table>
<thead>
<tr>
<th>Dim.</th>
<th>#Maximal Cones</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>768</td>
</tr>
<tr>
<td>3</td>
<td>114,432</td>
</tr>
<tr>
<td>4</td>
<td>1,169,792</td>
</tr>
<tr>
<td>5</td>
<td>1,007,616</td>
</tr>
<tr>
<td>6</td>
<td>2,443,136</td>
</tr>
<tr>
<td>7</td>
<td>4,743,904</td>
</tr>
<tr>
<td>8</td>
<td>109,920</td>
</tr>
</tbody>
</table>

TABLE V: Number of maximal cones in the tropical prevariety of cyclic-16, grouped by dimension.
3.4.2.3 4×4 Minors of a 5×5 Matrix

In (25), the authors pose the question of whether or not the 4×4 minors of a 5×5 matrix form a tropical basis. It was answered in the affirmative in (23), where one proof strategy required computing the tropical prevariety defined by the 4×4 minors of the following matrix:

\[
\begin{pmatrix}
  x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\
  x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\
  x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\
  x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\
  x_{51} & x_{52} & x_{53} & x_{54} & x_{55}
\end{pmatrix}.
\]

In (23), the 4×4 minors tropical prevariety was computed in two weeks, using four threads, or eight weeks of computation time if run linearly. They also exploited symmetry of the problem, which reduced the computation time by an expected factor between 2-10x.

DynamicPrevariety completed this computation in under ten hours using twenty threads. However, these two trials cannot fairly be compared, as processor speeds have increased in the eight years since the computation was run in (23).

3.4.2.4 Tropicalization of del Pezzo Surfaces

In (75), Q. Ren, Shaw, and Sturmfels present the conjecture that a set of 270 equations in 27 variables with 6 parameters forms a tropical basis. This set of equations is obtained from \( \mathbb{P}^2 \) by blowing up 6 points in general position.
In an ongoing work with Paul Görlach and Yue Ren, we disproved this conjecture through computing the tropical prevariety of a special instance of this ideal. We specialized the ideal, picking explicit values for the parameters, which transformed the generators of the ideal into a set of 270 trinomials in 27 unknowns. DynamicPrevariety was able to compute the tropical prevariety of this set in thirty seconds on an ordinary laptop, while it had previously been computationally infeasible.

### 3.4.3 Using DynamicPrevariety

Like most software in scientific computing, DynamicPrevariety is run from the command line. It reads in input text files of support sets and outputs text files containing the tropical prevariety in the same directory as it is run. We now present an example of computing the tropical prevariety of the cyclic-6 roots problem. Run the command:

```
./dynamicprevariety ./examples/cyclic/cyclic6
```

where the contents of the file `./examples/cyclic/cyclic6` are the support of the polynomial system in this format:

```
[[1,0,0,0,0,0], [0,1,0,0,0,0], [0,0,1,0,0,0], [0,0,0,1,0,0], [0,0,0,0,1,0], [0,0,0,0,0,1]]
[[1,1,0,0,0,0], [0,1,1,0,0,0], [0,0,1,1,0,0], [0,0,0,1,1,0], [1,0,0,0,0,1], [0,0,0,0,1,1]]
[[1,1,1,0,0,0], [0,1,1,1,0,0], [0,0,1,1,1,0], [1,1,0,0,0,1], [1,0,0,0,1,1], [0,0,0,1,1,1]]
[[1,1,1,1,0,0], [0,1,1,1,1,0], [1,1,0,0,1,1], [1,0,0,1,1,1], [0,0,1,1,1,1]]
[[1,1,1,1,1,0], [1,1,1,1,0,1], [1,1,0,1,1,1], [1,0,1,1,1,1], [0,1,1,1,1,1]]
[[1,1,1,1,1,1], [0,0,0,0,0,0]]
```

When DynamicPrevariety terminates, it will output the following file, up to a permutation of the rays:
------ Rays ------
4: {-2,1,1,-2,1,1,}
2: {-1,-1,2,-1,-1,2,}
0: {-1,1,-1,1,-1,1,}
5: {-1,2,-1,-1,2,-1,}
3: {1,-2,1,1,-2,1,}
1: {1,-1,1,-1,1,-1,}
6: {1,1,-2,1,1,-2,}
7: {2,-1,-1,2,-1,-1,}

------ Cones of dimension 1 ------
{0,}
{1,}

------ Cones of dimension 2 ------
{2,3,}
{2,4,}
{5,6,}
{4,5,}
{3,7,}
{6,7,}

------ F-vector ------
{ 2, 6 }

This file has sufficient information to reconstruct the tropical prevariety while being both terse and human-readable.

In addition to this basic usage, there are also a variety of options that DynamicPrevariety implements.

- `-t x` allows the user to run DynamicPrevariety, with x threads. If not specified, one thread is used.
- `-l/-u` sets DynamicPrevariety to find the intersection of the tropical prevariety with the open lower/upper half space.
- `-d x` returns the first cone found of the tropical prevariety that is of dimension > x.
- `-h` returns only the highest dimensional maximal cones in the tropical prevariety.
• \( + \) requires that the vertices in the support be provided with signs (+ or -), such that there is exactly one (+) per support set. When this option is used, only the edges that have opposite signs will be considered when computing the tropical prevariety. This is useful in certain applications, such as [5].

• \(-gfan\) changes the output format of DynamicPrevariety to that of Gfan, with a few caveats. LINEALITY_SPACE and ORTH_LINEALITY_SPACE are always empty. SIMPLICIAL and PURE are always 0. CONES is equal to MAXIMAL_CONES, instead of the full cone structure.

• \(-polymake\) changes the output format of DynamicPrevariety to the XML format used by Polymake [37, 38].

• \(-v\) increases the verbosity of the program, giving some feedback during runtime.

In addition to these options, we have also written a Sage interface to DynamicPrevariety that takes as input either a list of polynomials or an ideal and returns the cones in the tropical prevariety as well as the generating rays.

```
sage: attach("dynamic_prevariety_interface.sage")
sage: R.<x1,x2,x3,x4,x5,x6> = QQ[]
sage: cyclic6 = sage.rings.ideal.Cyclic(R,6)
sage: TropicalPrevariety(cyclic6)
([1-d cone in 6-d lattice N,
  1-d cone in 6-d lattice N,
  2-d cone in 6-d lattice N,
  2-d cone in 6-d lattice N,
  2-d cone in 6-d lattice N,
  2-d cone in 6-d lattice N,
  2-d cone in 6-d lattice N,
  2-d cone in 6-d lattice N],
  [[-2, 1, 1, -2, 1, 1],
```
We have also made DynamicPrevariety available to try in the cloud via the website

www.phcpack.org.

3.5 Future Directions

Though we have greatly improved the ability to compute tropical prevarieties, there are still more improvements that can be made. A low hanging fruit is integrating both algorithms from Section 3.2 and Section 3.3 into a single software package. In general, the dynamic enumeration algorithm is better, but when there are exactly two tropical hypersurfaces, walking the associated Newton polytopes is always more efficient. A large improvement could come from developing a dedicated linear programming library that does not continually allocate while it runs. Another improvement could be in automatically exploiting symmetry, as many interesting problems have some kind of symmetry.

An interesting venue for future research is in exploring what we call dynamic decomposition. In all of our algorithms that involve half open cones, we divide the fan into half open cones once at the beginning of the algorithm. We do this in an arbitrary way through picking a random ray, but this can result in inefficiencies as displayed in Figure 13. It seems possible that we could divide the fans into half open cones repeatedly as the algorithm progresses, and in doing so, we would always divide the fans in the optimal way.
Figure 13: A cone and a fan made up of half open cones. If the cone is intersected with the fan, three cones will result. If the fan was split into half open cones differently, the intersection could result in only one cone.
CHAPTER 4

SOLVING POLYNOMIAL SYSTEMS

WITH MONODROMY

4.1 Problem Statement

Numerically solving systems of polynomial equations is a mature field \cite{72, 79}, but there are still challenges in solving certain types of systems. For sparse polynomial systems, using polyhedral methods to compute all solutions is widely considered to be the optimal method \cite{85, 50}; however, it does not perform well in practice when we seek to solve polynomial systems where the root count is below the mixed volume. This chapter introduces our framework for solving polynomial systems via monodromy and homotopy continuation \cite{27}. This framework can be utilized to solve any polynomial system, but it outperforms all other methods on certain parametrized polynomial systems.

Problem Statement: Find all solutions of a generic square polynomial system $F(x)_p$ in a family of polynomial systems with parametric coefficients.

We begin this chapter with more background and by defining a series of terms that describe how to represent the problem as a graph. We will then show how this is useful by giving several algorithms and showing experimental results.
Let $F(x)_p$ be a square polynomial system of $n$ variables, with parameters $p = (p_1, \ldots, p_m) \in \mathbb{C}^m$. Let $B$ be the associated base space, by which we mean the parametrized linear variety of systems. We think of it as the image of an affine linear map $\varphi : p \mapsto F(x)_p$ from the parameter space to the space of systems.

We assume the structure of our family is such that the projection $\pi$ from the solution variety

$$V = \{(F_p, x) \in B \times \mathbb{C}^n \mid F_p(x) = 0\} \quad (4.1)$$

to $B$ gives us a branched covering, that is, that the fiber $\pi^{-1}(F_p)$ is finite of the same cardinality for a generically chosen $p$. The discriminant variety $D$ is the set of systems in the base space with nongeneric fibers.

The fundamental group $\pi_1(B \setminus D)$ as a set consists of loops, or paths in $B \setminus D$ that start and end at a fixed $p \in B \setminus D$. Each loop induces a permutation of the fiber $\pi^{-1}(F(x)_p)$, which is referred to as a monodromy action.

To reword the problem statement, our goal is to find the fiber of one generic system in the family. Our method is to find one pair $(p_0, x_0) \in V$ and use the monodromy action on the fiber $\pi^{-1}(F(x)_{p_0})$ to find its points.
4.2.1 Monodromy

We now will briefly introduce some facts about monodromy groups of branched coverings.

Consider a system $\mathbf{F}(x)_p \in B \setminus D$ and a loop $\tau$ without branch points based at $\mathbf{F}(x)_p$. This means that $\tau$ is a continuous path

$$\tau : [0, 1] \to B \setminus D \quad (4.2)$$

such that $\tau(0) = \tau(1) = \mathbf{F}(x)_p$. Let $x_i$ be a point in the fiber $\pi^{-1}(\mathbf{F}(x)_p)$ with $d$ points $x_1, x_2, \ldots, x_d$. Since $\pi$ is a covering map, the pair $(\tau, x_i)$ corresponds to a unique lifting $\tilde{\tau}_i$, a path

$$\tilde{\tau}_i : [0, 1] \to V \quad (4.3)$$

such that $\tilde{\tau}_i(0) = x_i$ and $\tilde{\tau}_i(1) = x_j$ for some $1 \leq j \leq d$. Note that the reversal of $\tau$ and $x_j$ lift to a reversal of $\tilde{\tau}_i$. The loop $\tau$ induces a permutation of the set $\pi^{-1}(\mathbf{F}(x)_p)$. We have a group homomorphism

$$\varphi : \pi_1(B \setminus D, \mathbf{F}(x)_p) \to S_d \quad (4.4)$$

whose domain is the usual fundamental group of $B \setminus D$ based at $\mathbf{F}(x)_p$. The image of $\varphi$ is the monodromy group associated to $\pi^{-1}(\mathbf{F}(x)_p)$. The monodromy group acts on the fiber $\pi^{-1}(\mathbf{F}(x)_p)$ by permuting the solutions of $\mathbf{F}(x)_p$. 
4.3 Graph of Homotopies

In order to find all of the isolated solutions of $F(x)_p$, we represent the problem as a finite undirected graph $G$. Let $E(G)$ and $N(G)$ denote the edge set and node set of $G$, respectively. Every node $n \in N(G)$ is associated to a specific $p_i$ chosen randomly. An edge $e \in E(G)$ that connects $n_1, n_2 \in N(G)$ is labeled with $\gamma_1, \gamma_2 \in \mathbb{C}$, and represents the linear homotopy connecting $\gamma_1 F(x)_{p_1}$ and $\gamma_2 F(x)_{p_2}$. We assume that both $p_i$ and $\gamma_i$ are chosen such that the segments do not intersect the discriminant variety. Choosing these at random satisfies this assumption, since the exceptional set of choices where such intersections happen is contained in a real Zariski closed set (79, Lemma 7.1.3).

We allow there to be multiple edges between two distinct nodes but no loops, since that would lead to trivial homotopies. However, for a graph $G$ to have the potential to be useful, it must not be acyclic. Given this basic setup, we describe some of the basic components of our algorithms:

- For each $n_i$, we keep track of the known solutions at the node.
- For each edge $e$ connecting $n_i$ and $n_j$, we store the correspondences (i.e. we store that $e$ maps the $k^{th}$ solution of $n_i$ to the $l^{th}$ solution of $n_j$). At the initialization of the algorithm, we will not know any correspondences.
- The graphs are initialized with what we call a seed pair $(p_0, x_0)$ by picking $x_0 \in \mathbb{C}^n$ at random and choosing $p_0$ to be a generic solution of the linear system $F(x_0)_p = 0$. In this way, we know a single solution on a single node when we begin.
As an example of our framework, suppose we want to find the roots of a generic univariate cubic polynomial. Writing it as

\[ c_1 x^3 + c_2 x^2 + c_3 x + c_4, \]  

we could set up \( G \) for three values \( p_1, p_2, p_3 \in \mathbb{C}^4 \) of the coefficients, as in Figure 14 where we assume we know one root \( \alpha_1 \) for \( p_1 \). By tracking this solution between the three nodes, we expect to recover all three solutions at one of the nodes. One possible outcome is shown in Figure 15.

Figure 14: A basic graph of homotopies, with three nodes and one edge between each pair of nodes. Dashed lines represent correspondences that have not yet been discovered.
At this point, an algorithm can be imagined which would find all solutions of \( \mathbf{F}(\mathbf{x})_{p_1} \) (visualized in Figure 15):

1. seed \( \mathbf{F}(\mathbf{x})_{p_1} \) with known solution \( \alpha_1 \)
2. track \( \alpha_1 \) towards \( \mathbf{F}(\mathbf{x})_{p_2} \) and discover \( \beta_2 \)
3. track \( \beta_2 \) towards \( \mathbf{F}(\mathbf{x})_{p_3} \) and discover \( \lambda_2 \)
4. track \( \lambda_2 \) back to \( \mathbf{F}(\mathbf{x})_{p_1} \) and discover \( \alpha_3 \)
5. track \( \alpha_1 \) towards \( \mathbf{F}(\mathbf{x})_{p_3} \) and discover \( \lambda_3 \)
6. track \( \lambda_3 \) towards \( \mathbf{F}(\mathbf{x})_{p_2} \) and discover \( \beta_3 \)
7. track \( \beta_3 \) back to \( \mathbf{F}(\mathbf{x})_{p_1} \) and discover \( \alpha_2 \)

Note that it is not necessary to complete all correspondences in the graph of homotopies in order to recover all solutions. Doing so would require tracking nine paths, while the hypothetical run we describe only requires six path tracks to find a fiber.

4.4 Strategies, Algorithms and Potential Functions

We now work to make the intuition we just described rigorous. We present both static and dynamic algorithms for populating all the solutions on some node in \( \mathcal{G} \). We call an algorithm \textit{static} if the homotopy graph is created at the beginning of the algorithm and remains unchanged throughout; we call an algorithm \textit{dynamic} if the homotopy graph could change during the run of the algorithm, i.e. gain or lose nodes and/or edges.
4.4.1 Naive Dynamic Strategy

This strategy is not one of our contributions, but we include it as background. Existing implementations of numerical irreducible decomposition in Bertini \cite{Bertini}, PHCpack \cite{PHCpack}, and NumericalAlgebraicGeometry for Macaulay2 \cite{NumericalAlgebraicGeometry} that use monodromy are driven by some version of what we call the naive dynamic strategy. This algorithm is presented in Algorithm \ref{alg:naive-dynamic}.

Algorithm \ref{alg:naive-dynamic} can be described using the language of our framework. Instantiate $\mathcal{G}$ with a seed pair $(p_0, x_0)$. Create a new node, track to and from it. Repeat until all solutions are known on the initial node.
Algorithm 12

Input: $F_{p_0}$ with solution $x_0$, $d$ the known root count.
Output: All $d$ solutions of $F_{p_0}$.

while $(\text{Solutions of } F_{p_0}) < d$ do
    Pick random $p_i, \gamma_{i_1}, \gamma_{i_2}$.
    Track all known solutions of $F_{p_0}$ to $F_{p_i}$ along $\gamma_{i_1}$.
    Track all known solutions of $F_{p_i}$ back to $F_{p_0}$ along $\gamma_{i_2}$.
    Combine possibly new solutions of $F_{p_0}$ with known solutions of $F_{p_0}$.
end while

This strategy will find all the solutions for $F_{p_0}$, but it is natural to ask how quickly the solutions will be found. For the first trip away from and then back to $F_{p_0}$, the probability of finding a new solution equals $\frac{d-1}{d}$, where $d$ is the total number of solutions of $F_{p_0}$. As additional solutions are discovered on $F_{p_0}$, the probability of finding any new solutions decreases.

Finding the expected number of cycles through the while loop in Algorithm 12 is equivalent to the coupon collector’s problem. The number of iterations is expected to be $d \ell(d)$ where $

\ell(d) := \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{d}$.

4.4.2 Static Graph Strategy

In contrast to the naive dynamic strategy, our framework is most powerful when we reuse the nodes and edges of the graph. In a static strategy, we fix a graph and discover solutions according to Algorithm 13.

We remain somewhat vague on the details of Algorithm 13 but we will become more explicit in Section 4.4.4. We exclude a discussion on one of the results of (27, Section 4) which describes the computational complexity of this framework. We prove that in our framework where edges
Algorithm 13 Static graph strategy

Input:
- A graph of homotopies $\mathcal{G}$, where at least one solution is known.
- A stopping criterion $\text{stop}$.

Output: $F(x)_{p_j}$ and a set of solutions to it.

\begin{algorithm}
\begin{algorithmic}
\State \textbf{while} there exists an edge on $\mathcal{G}$ such that it has trackable solutions \textbf{do}
\State \hspace{1em} Choose such an edge $e$, connecting nodes $n_i$ and $n_j$.
\State \hspace{1em} Let $S$ be a nonempty subset of the set of solutions not yet tracked on $e$.
\State \hspace{1em} Track $S$ from $n_i$ to $n_j$.
\State \hspace{1em} Integrate the solutions found with the previously known solutions on $n_j$
\State \hspace{1em} \textbf{if} the criterion $\text{stop}$ is satisfied \textbf{then}
\State \hspace{2em} \textbf{return} $F(x)_{p_j}$ associated to $n_j$ and solutions of $n_j$
\State \hspace{1em} \textbf{end if}$\quad$\text{5:}
\State \hspace{1em} \textbf{end while}$\quad$\text{10:}
\State Let $n_j$ be the node with the most known solutions.
\State \textbf{return} $F(x)_{p_j}$ associated to $n_j$ and solutions of $n_j$
\end{algorithmic}
\end{algorithm}

and nodes are reused, the expected number of path tracks is linear in the number of solutions, and we also show that result experimentally.

4.4.3 Incremental Dynamic Graph Strategy

We introduce a more sophisticated dynamic strategy, that uses the static graph strategy just described as its main routine. This dynamic strategy amounts to augmenting the graph once one of the above “static” criteria terminates Algorithm 13 for the current graph.

A simple dynamic stopping criterion is to fix the number of augmentation steps that the algorithm is allowed to make without increasing the solution count.

4.4.4 Specializing the Algorithms

Algorithm 13 can be specialized in three main ways. We can:
Algorithm 14 Dynamic graph strategy

**Input:**
- A graph $\mathcal{G}$ as in Algorithm 13 along with its stopping criterion.
- A dynamic stopping criterion $\text{dynamic\_stop}$.
- An augmenting procedure $\text{augment}$.

**Output:** $F(x)_{p_j}$ and a set of solutions to it.

```plaintext
while True do
    $(F(x)_{p_j}, \text{solution set}) = \text{perform Algorithm 13 on } \mathcal{G}$
    if the criterion $\text{dynamic\_stop}$ is satisfied then
        return $(F(x)_{p_j}, \text{solution set})$
    end if
    $\mathcal{G} := \text{augment}(\mathcal{G})$
end while
```

1. choose the graph $\mathcal{G}$,

2. pick the stopping criterion $\text{stop}$,

3. define the strategy for picking the next edge to track $e$.

We address each of these choices in this section.

### 4.4.4.1 Basic Static Graphs

We mention two static graph layouts that we have found to be useful, with depictions of them in Figure 16.

**flowerGraph(#n,#e):** This graph consists of a central node $n_0$ and #n additional nodes. Each node is connected to the central node by #e edges. We call it the flower graph because of its appearance. Note that the layout of this graph is similar to that generated by the naive dynamic strategy.
**completeGraph(#n,#e):** As the name suggests, this layout is a complete graph with #n nodes and #e edges connecting each pair of nodes.

![Diagram of completeGraph(6,1) and flowerGraph(4,2) layouts](image)

Figure 16: Sketches of the flowerGraph(4,2) layout and completeGraph(6,1) layout.

### 4.4.4.2 Stopping Criteria

There are two natural situations where we must determine when the algorithm should terminate: when we know the target number of solutions and when we do not know the target number of solutions.

When we are in the first situation, the stopping criterion is simply to check if the number of solutions on a node is equal to the known root count. When our system is a generic sparse system of fixed monomial support, we can compute a mixed volume and use this as the stopping criterion [9]. Note that the computation of the mixed volume can be done entirely
independently from the running of our static graph strategy. In general, computing a mixed volume is much faster than running Algorithm [13].

A stopping criterion without a known number of solutions usually will depend on heuristics.

1. A simple heuristic is saturating all of the correspondences along all of the edges. Conceptually, this exhausts the graph, and if the graph is sufficiently large, all solutions will be recovered at all nodes.

2. A more efficient heuristic is stabilization. The algorithm exits when no new solutions are found in a fixed number of iterations. This is useful in practice, because it will avoid saturating all of the correspondences.

There are certain cases where it is possible to use the trace test as a stopping criterion ([78], [63]). The trace test provides a more rigorous check of whether or not all solutions have been found, but it is beyond the scope of this work.

4.4.4.3 Edge Selection Strategy

We introduce four methods for selecting the edge $e$ in Algorithm [13]. To make the selection, we must pick a potential function, which we evaluate on each edge. We pick the edge with the highest potential function. Let $e = n_i \xrightarrow{(\gamma_i, \gamma_j)} n_j$ be a directed edge from $n_i$ to $n_j$. Let $|n|$ be the number of known solutions at $n$ and let $|e|$ be the number of known correspondences on the edge.

potentialRandom evaluates the potential of every edge to zero. Thus, every edge is equally likely to be chosen.
potentialLowerBound is the minimal number of new solutions guaranteed to be discovered by picking $e$. It equals $\max(|n_i| - |n_j|, 0)$.

potentialE represents the expected number of new solutions obtained by tracking one path along $e$. It is the ratio $\frac{d - |n_j|}{d - |e|}$ of undiscovered points among all unmatched points.

potentialWeightTowardCompleteNode is equal to $\text{potentialE} + |n_j|$. The intuition behind this function is that it will seek to fill up a single node, while leaving all other nodes minimally populated.

Note that some of these potential functions are only possible when the root count is known at the beginning of the computation. Some of the potential functions we have just described are greedy. From our experiments [Table VI and Table VII], it is evident that they will evaluate edges differently, resulting in differences in performance.

4.5 Implementation

To test our algorithms, we implemented the package MonodromySolver in Macaulay2 (41), which calls the native Macaulay2 path tracker (62). The package has been included in the Macaulay2 distribution since version 1.10, and the source code is available at (28).

The main function in the package is monodromySolve, which implements Algorithms 13 and 14. The following example illustrates using MonodromySolver as a blackbox, not specifying any of the specializations we have described:

```plaintext
i1 : needsPackage "MonodromySolver"
o1 = MonodromySolver
```
For the advanced user, we provide the ability to experiment with all of the features we have described. The following example continues the previous example, and it demonstrates some of the optional inputs that monodromySolve accepts:

```
\texttt{i8 : (p0,x0) = createSeedPair polys;}
\texttt{i9 : (N,npaths) = monodromySolve(polys,p0,{x0},
SelectEdgeAndDirection=>selectBestEdgeAndDirection,
Potential=>potentialE,
NumberOfNodes=>3,
NumberOfEdges=>3,)
```
GraphInitFunction => flowerGraphInit,
AugmentGraphFunction => flowerGraphAugment,
AugmentEdgeCount => 1,
AugmentNumberOfRepeats => 10);

i10 : npaths

o10 = 14

The output of \texttt{monodromySolve} is a tuple that contains a \texttt{HomotopyNode} and the total number of paths tracked. The \texttt{HomotopyNode} is a \texttt{Macaulay2} object with information about the system, which includes the output promised by Algorithm 13. Note that the number of paths required is lower when the user strategically sets optional parameters.

The overhead of managing the data structures is supposed to be negligible compared to the cost of tracking paths. However, since our implementation uses the interpreted language of \texttt{Macaulay2} for the bookkeeping, this overhead can be sizable (we found it to be up to 10\% for large examples). Because of this, most of our experiments focus on measuring the number of tracked paths as a proxy for computational complexity.

The mixed volume can be used as an expected solution count when $F(x)_p$ is a sparse system. Our current implementation calls \texttt{PHCpack} to compute the mixed volume, which uses a modified version of \texttt{MixedVol}. There have been several recent advances in computing mixed volumes, namely in the software packages \texttt{pss5} (68) and \texttt{Gfanlib} (55). Our implementation could call either of them, but we have not found the computation of a mixed volume to be a bottleneck in our software.
4.6 Experiments

In this section, we analyze how our framework does on its own, and then we proceed to compare it to the universe of other solvers.

4.6.1 Sparse Polynomial System

For our canonical sparse polynomial system, we again use the cyclic-$n$ roots family. We study a modified version of these systems, where the coefficients are randomized, and we seek solutions in $(\mathbb{C} \setminus \{0\})^n$. The solution count can be computed as the mixed volume of the polynomial system, so we use that natural stopping criterion. For cyclic-7, this bound is 924.

Table VI and Table VII contain averages of experimental data from running twenty trials of Algorithm 13 on cyclic-7. The main measurement reported is the average number of paths tracked, as the unit of work for our algorithm is performing a single homotopy path track. The experiments were performed with 10 different graph layouts and 3 edge selection strategies. We say that a graph was successful if it found all of the solutions of the polynomial system. The success rate in the tables corresponds to the percentage of time a graph was successful.

<table>
<thead>
<tr>
<th>(#nodes-1, edge multiplicity)</th>
<th>(3,2)</th>
<th>(4,2)</th>
<th>(5,2)</th>
<th>(3,3)</th>
<th>(4,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success rate</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>potentialRandom</td>
<td>5119</td>
<td>6341</td>
<td>7544</td>
<td>6100</td>
<td>7067</td>
</tr>
<tr>
<td>potentialLowerBound</td>
<td>5252</td>
<td>6738</td>
<td>8086</td>
<td>6242</td>
<td>7886</td>
</tr>
<tr>
<td>potentialE</td>
<td>4551</td>
<td>5626</td>
<td>6355</td>
<td>4698</td>
<td>5674</td>
</tr>
</tbody>
</table>

TABLE VI: Cyclic-7 experimental results for the flowerGraph strategy.
We point out several basic observations from Table VI and Table VII. For a fixed polynomial system, as the size of the graph increases, the probability of finding all solutions also increases. However, this also correlates with dramatically more path tracks, so this added certainty comes at a cost. Again, this could be mitigated if a dynamic strategy was used, and the starting graph was small. Finally, note that the greedy potentialE outperforms the other tested potential functions.

4.6.2 Success Rate

We investigate the success rate of Algorithm 13 for the Katsura family parametrized by \( n \) with fixed support and generically chosen coefficients \( \{16, 60\} \). Table VIII and Table IX contain the percentage of successes from 500 runs with distinct random seeds.

We note that as the first Betti number increases, the probability of success also improves. Furthermore, as the number of solutions increases, the probability of a transitive action also decreases, so again, the success rate increases.

<table>
<thead>
<tr>
<th>(#nodes, edge multiplicity)</th>
<th>(2,3)</th>
<th>(2,4)</th>
<th>(2,5)</th>
<th>(3,2)</th>
<th>(4,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success rate</td>
<td>65%</td>
<td>80%</td>
<td>90%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>potentialRandom</td>
<td>2728</td>
<td>3296</td>
<td>3947</td>
<td>4805</td>
<td>5165</td>
</tr>
<tr>
<td>potentialLowerBound</td>
<td>2727</td>
<td>3394</td>
<td>3821</td>
<td>4688</td>
<td>5140</td>
</tr>
<tr>
<td>potentialE</td>
<td>2692</td>
<td>2964</td>
<td>2957</td>
<td>3886</td>
<td>4380</td>
</tr>
</tbody>
</table>

TABLE VII: Cyclic-7 experimental results for the completeGraph strategy.
### 4.6.3 Comparison to Other Solvers

All timings appearing in this section were done with a single thread on the same machine, using MonodromySolver. In this section, we demonstrate the practicality of our approach in two situations: when the actual solution count is less than the mixed volume and when it is equal to the mixed volume.
Our core contribution is that this framework improves our computational ability in polynomial systems where the solution count is significantly smaller than the mixed volume. Existing blackbox methods that expect to attain the mixed volume number of solutions likely spend much more time in computation than our approach.

In Table X, we collect timings on several challenging examples mentioned in recent literature where smaller solution counts are known. This provides us with rigorous test cases for our heuristic stopping criterion. The first system in the table is that of the wnt signaling pathway reaction network described in (42). The others come from the problem of computing the degree of $SO(n)$, the special orthogonal group, as a variety (18).

For these examples, we chose small graphs with $\beta_1 \leq 4$, and we used the random edge selection strategy. Since we did not know the root count ahead of time, we used the stabilization stopping criterion. Note that while the blackbox solver of \texttt{PHCpack} performs polyhedral homotopy continuation, \texttt{Bertini} relies on the equation-by-equation technique of regeneration. The latter may be faster than the former in certain cases, which Table X shows.

<table>
<thead>
<tr>
<th>problem</th>
<th>wnt</th>
<th>$SO(4)$</th>
<th>$SO(5)$</th>
<th>$SO(6)$</th>
<th>$SO(7)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution count</td>
<td>9</td>
<td>40</td>
<td>384</td>
<td>4768</td>
<td>111616</td>
</tr>
<tr>
<td>MonodromySolver</td>
<td>0.52</td>
<td>4</td>
<td>23</td>
<td>528</td>
<td>42791</td>
</tr>
<tr>
<td>Bertini</td>
<td>42</td>
<td>81</td>
<td>10605</td>
<td>out of memory</td>
<td></td>
</tr>
<tr>
<td>\texttt{PHCpack}</td>
<td>862</td>
<td>103</td>
<td>&gt; one day</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE X: Examples with solution count smaller than mixed volume. Timings in seconds.
When the solution count is equal to the mixed volume, our framework is a viable alternative to polyhedral homotopy solvers, since the number of paths we track is linear in the number of solutions. We present timings on several large benchmark problems in Table XI. We again used graphs with small Betti numbers, and had no issues with success rates.

**Comparison to polyhedral solvers:** Unlike the other software systems we mention, HOM4PS2 is not open source. HOM4PS2 may use just-in-time compilation of straight-line programs to evaluate polynomial system. If this is done, it speeds up computations considerably. PHCpack does not use this technique and neither does our software. However, experiments done by others in Macaulay2 suggest that there could be at least a 10x speedup if MonodromySolver did use just-in-time compilation.

<table>
<thead>
<tr>
<th>problem</th>
<th>cyclic-10</th>
<th>cyclic-11</th>
<th>noon-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixed volume</td>
<td>35940</td>
<td>184756</td>
<td>59029</td>
</tr>
<tr>
<td>completeGraph(2,3)</td>
<td>610</td>
<td>7747</td>
<td>failed</td>
</tr>
<tr>
<td></td>
<td>(107820 paths)</td>
<td>(540155 paths)</td>
<td>(59001 solutions)</td>
</tr>
<tr>
<td>completeGraph(2,4)</td>
<td>740</td>
<td>8450</td>
<td>935</td>
</tr>
<tr>
<td></td>
<td>(129910 paths)</td>
<td>(737432 paths)</td>
<td>(236051 paths)</td>
</tr>
<tr>
<td>PHCpack</td>
<td>538</td>
<td>4256</td>
<td>751</td>
</tr>
<tr>
<td>HOM4PS2</td>
<td>62</td>
<td>410</td>
<td>120</td>
</tr>
</tbody>
</table>

**TABLE XI:** Examples with solution count equal to mixed volume. Timings in seconds.

**Comparison to non-polyhedral solvers:** On the examples in Table XI, we also ran the blackbox solvers of Bertini and NumericalAlgebraicGeometry, which use total-degree homotopies.
Both were able to finish noon-10 with timings similar to the table, but all other problems took longer than a day. This is because the mixed volume of noon-10 is only slightly sharper than the Bézout bound, while that is not true for the cyclic-\(n\) roots problems.

**Comparison to naive dynamic strategy:** When compared to the naive dynamic strategy we originally described, our framework is only worse in the aspect of memory consumption. In this kind of problem, memory consumption is driven entirely by storing the floating point representations of the solutions. For a problem with \(d\) solutions, the naive approach stores no more than \(2d\) points. The number of points our framework stores is up to (though usually much less than) \(d\) times the number of nodes.

More importantly, the number of paths tracked is significantly lower in our framework than in the naive dynamic strategy. For example, the naive strategy tracks about 7500 paths on average for cyclic-7. In Table VI and Table VII, every option ran far fewer than 7500 paths. However, it is possible that a poorly chosen graph of homotopies could lead to tracking more paths in our framework. One way this could be done would be if a huge graph was initialized with a random potential function.

### 4.7 Parallel Monodromy

We have worked on developing a parallel version of using monodromy in conjunction with homotopy continuation methods, similar in some ways to [64]. Traditional homotopy continuation methods are pleasingly parallel: no path track impacts any other path track, so all can occur simultaneously. However, in our framework, treating all the path tracks as independent can only be done when the random potential function is used. If a greedy potential function
is used, once a single path track has been launched, a subsequent path track must also be launched, but it is not obvious how its potential can be evaluated before the first path track terminates. We give a brief overview of our method to do just that, but for more details see [14].

When a path tracker in progress completes, that tracker will either

- find or not find a new solution, and
- create or not create a new correspondence.

Since the path track is in progress, we have what we call a fuzzy graph, because we do not know what state our graph will be in when the current path track completes. Our goal then is to estimate how many solutions each node will have when the current path track terminates.

Here is an intuition for what we argue is the correct way to estimate how many solutions are at each node when we leave the fuzzy state:

1. Enumerate all possible graphs that could result from the current fuzzy state.
2. Average the number of known solutions at each node in all of these possible future graphs.
3. Apply the potential function to this averaged graph.

This is computationally feasible when there is only a single path track in progress, but as the number of threads increases, it quickly becomes impossible to do.

Because of this difficulty, we introduce and prove the equivalence of an different method for determining what state the graph will be in when it exits a fuzzy state. Every time a path track launches, we say that the node that it is tracking towards now has a fractional number of solutions: the real number of solutions added to the expected number of new solutions that this
path track will yield. These fractional solution counts can be used to evaluate potential functions as we did before with integer solution counts. We demonstrate a way to compute these values recursively, requiring a negligible amount of computation for each additional path launch and finish. We have implemented this method in a C++ simulator, and initial experiments show that this leads to near linear speedups for large polynomial systems with large numbers of threads.

4.8 Future Direction

This work represents a first foray into solving polynomial systems using homotopy continuation and monodromy, but much remains to be studied. We hope to analyze how robust this framework is to numerical failures of the path tracker. Is it beneficial to set the tracker to be more aggressive and have more path failures, or is it optimal to minimize the number of failures? If we know that we are going to have failures in our path tracks, how can we incorporate that information into a potential function?

On the practical side, we consider our implementation of MonodromySolver only a proof of concept. An algorithm like this would benefit from a low-level implementation, with parallelism implemented in it as well. It would also be interesting to attempt to integrate this framework with polyhedral methods or regeneration. A hybrid approach like this may lead to an even better numerical blackbox solver.
CHAPTER 5

A COMPUTER ALGEBRA SYSTEM FOR R

5.1 Problem Statement

Macaulay2 is an open-source computer algebra system designed to perform computations in commutative algebra and algebraic geometry \cite{41}. Started in the early 2000s, the software has a large code base with many community members actively developing add-on packages. Macaulay2 also links to other major open source software in the mathematics community, such as PHCpack \cite{82, 13}, Normaliz \cite{19, 21, 20}, and 4ti2 \cite{1}, through a variety of interfaces.

One of the main benefits of Macaulay2 is its efficiency with large algebraic computations, especially Gröbner basis computations. Gröbner basis computations are at the center of many algorithms in computational algebraic geometry. Some of these computations take many hours and produce output of thousands of polynomials or polynomials with thousands of terms. Often, the Macaulay2 user will not be interested in the entire output, but only certain properties of the output. Macaulay2 allows the user to ask for only properties of interest to be returned, such as the dimension or the degree of the computation.

Neither rSymPy or mpoly meet the computational needs of those in the algebraic statistics community, because neither were designed with that community in mind. Because of this, for years those using algebraic statistical methods have been forced to leave R to manually run computations in Macaulay2 and then pull the results back into R. This error prone and tedious
process is a major barrier to entry to using algebraic statistics in \textit{R}. The problem is compounded by users needing to install \textit{Macaulay2}, which is not cross-platform, and needing to be familiar with the \textit{Macaulay2} language, which is syntactically very different from \textit{R}.

**Problem Statement:** Create an \textit{R} package that connects \textit{R} to \textit{Macaulay2} through a persistent back-end socket connection.

### 5.2 Basic Usage

To fill this need, we created \texttt{m2r}, an \textit{R} package that connects to an interactive \textit{Macaulay2} session. \texttt{m2r} is loaded like any other \textit{R} package:

\begin{verbatim}
R> library(m2r)
Loading required package: mpoly
Loading required package: stringr
M2 found in /usr/local/bin
\end{verbatim}

The first two lines of output indicate that \texttt{m2r} depends on \texttt{mpoly} and \texttt{stringr}. The packages \texttt{mpoly} and \texttt{stringr} manipulate and store multivariate polynomials and strings, respectively \cite{mpoly, stringr}. The third line indicates that \texttt{M2}, the \textit{Macaulay2} executable, was found on the user’s machine at the given path and that this version of \textit{Macaulay2} in that directory will be used for computations.

When loaded on a Unix-like machine, \texttt{m2r} looks for \texttt{M2} on the user’s machine by searching through \texttt{~/.bash_profile}, or if nonexistent, \texttt{~/.bashrc} and \texttt{~/.profile}. \texttt{m2r} stores the first place \texttt{M2} is found in the option \texttt{m2r\$m2\_path}. Note that \texttt{m2r} will not necessarily use whatever is on the user’s typical \texttt{PATH} variable because when \textit{R} makes \texttt{system()} calls, it does not load the user’s personal configuration files. If a different path is desired, the user can change this
option with the function `set_m2_path()`. When `m2r` is loaded, the M2 executable is found but not initialized. The actual initialization and subsequent connection to Macaulay2 by `m2r` takes place when R first calls a Macaulay2 function through `m2r`.

### 5.2.1 m2r Basics

The basic interface to Macaulay2 is provided by the `m2()` function. `m2()` accepts a string containing Macaulay2 code, sends it to Macaulay2 to be evaluated, and brings the output back into R. For example, like all computer algebra systems, Macaulay2 supports basic arithmetic:

```r
R> m2("1 + 1")
Starting M2...
done.
[1] "2"
```

Unlike most m2r functions, `m2()` does not parse the Macaulay2 output into an R data structure. This can be seen in the result above being a character and not a numeric, but it is even more evident when evaluating a floating point number:

```r
R> m2("1.2")
[1] ".12p53e1"
```

Parsing the output is a delicate task accomplished by the `m2_parse()` function:

```r
R> m2_parse(m2("1.2"))
[1] 1.2
```
For details on how `m2_parse()` works as a general Macaulay2 parser, see [58].

One of the main advantages of `m2r`'s implementation is that it provides a persistent connection to a Macaulay2 session running in the background. In early versions of `algstat`, Macaulay2 was accessible from R through intermediate script files; `algstat` saved user supplied Macaulay2 code to a temporary file, called `Macaulay2` in script mode to evaluate it, saved the output to another temporary file, and parsed the output back into R [57]. The major limitation of this scheme is that every computation and every variable created on the Macaulay2 side is lost once the call is complete. Unlike `algstat`, `m2r` allows for this kind of persistent connection to a Macaulay2 session, which can be demonstrated:

```r
R> m2("a = 1")
[1] "1"
R> m2("a")
[1] "1"
```

When not actively running code, the Macaulay2 session sits, listening for commands issued by R. The details of the connection are described in Section 5.4.

While the Macaulay2 session is live, it helps to have R-side functions that access it in a natural way. Because of this, just as there are functions such as `ls()` and `exists()` in R, `m2r` provides analogues for the background Macaulay2 session:

```r
R> m2_ls()
[1] "a"
```
m2_exists(c("a", "b"))
[1] TRUE FALSE
R> m2_getwd()
[1] "/home/jeff"

m2_ls() also accepts the argument all.names = TRUE, which gives a larger listing of the variables defined in the Macaulay2 session, much like ls(all.names = TRUE). These additional variables fall into two categories: output variables returned by Macaulay2 and m2r variables used to manage the connection. In Macaulay2, the output of each executed line of code is stored as a variable bound to the symbol o followed by the line number executed. For example, the output of the first executed line is o1. These are accessible through m2r as, for example, m2o1; however, since m2r's internal connection itself makes calls to Macaulay2, the numbering is unpredictable. Because of this, we exclude them from appearing in m2_ls(). The internal variables that m2r uses to manage the persistent connection to Macaulay2 are called m2rint* and shouldn't be accessed by the user.

5.2.2 m2r and Computational Algebraic Geometry

While the m2() function suffices at a basic level, it fails to provide a user-friendly interface to Macaulay2. m2r provides a number of wrapper functions and data structures that facilitate interacting with Macaulay2 in a way that is significantly more familiar to R users. In the remainder of this section, we demonstrate some of these functions in action. We begin with rings and ideals, the basic algebraic structures in commutative algebra, and conclude with the computation of Gröbner bases.
Polynomial rings can be created with the \texttt{ring()} function:

\begin{verbatim}
R> (R <- ring("t", "x", "y", "z", coefring = "QQ"))
\end{verbatim}

M2 Ring: \( \mathbb{Q}[t,x,y,z] \), grevlex order

Polynomial rings are comprised of two basic components: a collection of variables and a coefficient ring, usually a field. In \texttt{Macaulay2}, special key words are used to refer to commonly used coefficient rings: the integers \( \mathbb{Z} \) (\texttt{ZZ}), the rational numbers \( \mathbb{Q} \) (\texttt{QQ}), the real numbers \( \mathbb{R} \) (\texttt{RR}), and the complex numbers \( \mathbb{C} \) (\texttt{CC}). Polynomial rings and related algorithms can benefit from total orderings on their monomials. These can be supplied through \texttt{ring()}’s \texttt{order} argument, which by default sets \texttt{order = "grevlex"}, the graded reverse lexicographic order.

Ideals of rings can be specified with the \texttt{ideal()} function as follows:

\begin{verbatim}
R> (I <- ideal("t^4 - x", "t^3 - y", "t^2 - z"))
\end{verbatim}

M2 Ideal of ring \( \mathbb{Q}[t,x,y,z] \) (grevlex) with generators :
< \( t^4 \) - \( x \), \( t^3 \) - \( y \), \( t^2 \) - \( z \) >

They are defined relative to the last ring used that contains all the variables referenced. If no such ring exists, an error is returned. A common mistake is to try to reference a variable that cannot be scoped to a previously defined ring:

\begin{verbatim}
R> m2("u + 1")
Error: Macaulay2 Error!
\end{verbatim}

In situations where several rings have been used, the \texttt{use_ring()} function is useful for specifying which ring to use.
Gröbner bases of ideals are computed with \texttt{gb()}: 

\begin{verbatim}
R> gb(I)
z^2 - x
z t - y
-1 z x + y^2
-1 x + t y
-1 z y + x t
-1 z + t^2
\end{verbatim}

To be more user-friendly, \texttt{ideal()} and \texttt{gb()} are overloaded to accept many types of input, including \texttt{mpoly} and \texttt{mpolyList} objects. For example, instead of \texttt{gb()} working on an ideal object, it can work directly on a collection of polynomial strings:

\begin{verbatim}
R> gb("t^4 - x", "t^3 - y", "t^2 - z")
z^2 - x
z t - y
-1 z x + y^2
-1 x + t y
-1 z y + x t
-1 z + t^2
\end{verbatim}

Note the difference between the last two calls: \texttt{gb(I)} only took one argument, whereas \texttt{gb("t^4 - x","t^3 - y", "t^2 - z")} took three, but they performed the same task. Non-standard evaluation in \texttt{R} makes this possible (\cite{87} \cite{66}). While nonstandard evaluation is very convenient, it does have drawbacks. In particular, it tends to be hard to use functions that use nonstandard evaluation inside other functions. To alleviate this problem, each of \texttt{ring()}, \texttt{ideal()}, and \texttt{gb()} has a standard evaluation version that tends to be easier to program with and incorporate into packages. Following the \texttt{dplyr/tidyverse} naming convention (\cite{89}), these
functions have the same name followed by an underscore: \texttt{ring\_()}, \texttt{ideal\_()}, and \texttt{gb\_()}. To see the difference between standard and nonstandard evaluation, compare the previous \texttt{gb()} call, which depends on nonstandard evaluation, to this call to \texttt{gb\_()}, which uses standard evaluation:

\begin{verbatim}
R> polys <- c("t^4 - x", "t^3 - y", "t^2 - z")
R> gb_(polys, ring = R)

z^2 - x
z t - y
-1 z x + y^2
-1 x + t y
-1 z y + x t
-1 z + t^2
\end{verbatim}

Though the distinction is not as obvious, \texttt{gb(I)} and \texttt{gb\_(I)} both work and result in the same computation. The latter, however, is more appropriate for use inside packages.

Several other functions exist that aid in whatever one may want to do with ideals. For example, sums, products, and equality testing are all defined as methods of those \texttt{base} functions:

\begin{verbatim}
R> I <- ideal("x", "y")
R> J <- ideal("z")
R> I + J

M2 Ideal of ring QQ[t,x,y,z] (grevlex) with generators :
< x, y, z >

R> I * J

M2 Ideal of ring QQ[t,x,y,z] (grevlex) with generators :
< x z, z y >

R> I == J

[1] FALSE
\end{verbatim}
These can be combined with previous functions to great effect. For instance, it is simple to script a function to check whether an ideal is radical:

```r
R> is.radical <- function (I) I == radical(I)
R> is.radical(I)
[1] TRUE
```

### 5.2.3 More Examples of Macaulay2 Functionality

In addition to implementations of the basic Macaulay2 objects and algorithms of commutative algebra described above, m2r includes implementations of other algorithms that one might expect in a computer algebra system. For example, the prime decomposition of an integer can be computed with m2r’s `factor_n()`:

```r
R> (x <- 2^5 * 3^4 * 5^3 * 7^2 * 11^1)
[1] 174636000
R> (factors <- factor_n(x))

$prime
[1] 2 3 5 7 11

$power
[1] 5 4 3 2 1

R> str(factors)
List of 2
$ prime: int [1:5] 2 3 5 7 11
$ power: int [1:5] 5 4 3 2 1

R> gmp::factorize(x)
Big Integer (‘bigz’) object of length 15:
[1] 2 2 2 2 2 3 3 3 3 5 5 5 7 7 11
factor.n() is analogous to gmp’s factorize(), but it is significantly slower due to having to be passed to Macaulay2, computed, passed back, and parsed. On the other hand, conceptually m2r is factorizing the integer as an element of a ring, and it can do so more generally over other rings. Consequently, polynomials can be factored. The result is an mpolyList object of irreducible polynomials (the analogue to primes) and a vector of integers, as a list:

R> ring("x", "y", coeefring = "QQ")
M2 Ring: QQ[x,y], grevlex order
R> factor_poly("x^4 - y^4")
$factor
x - y
x + y
x^2 + y^2

$power
[1] 1 1 1

One could use this kind of connection, along with R’s random number generators, to experimentally obtain Monte Carlo answers to a number of mathematical questions. This kind of computation has applications in random algebraic geometry and commutative algebra.

A bit more interesting to statisticians may be the implementation of an algorithm to compute the Smith normal form of a matrix. The Smith normal form of a matrix M here refers to the decomposition of an integer matrix \( D = PMQ \), where \( D, P, \) and \( Q \) are integer matrices and \( D \) is diagonal. Both \( P \) and \( Q \) are unimodular matrices (their determinants are \( \pm 1 \)), so they are invertible. This is similar to a singular value decomposition for integer matrices.
```r
R> M <- matrix(c(
+   2,  4,  4,
+  -6,  6, 12,
+  10, -4, -16
+ ), nrow = 3, byrow = TRUE)
R>
R> mats <- snf(M)
R> P <- mats$P; D <- mats$D; Q <- mats$Q
R>
R> P %*% M %*% Q # = D
   [,1] [,2] [,3]
[1,]  12  0  0
[2,]   0  6  0
[3,]   0  0  2
R> solve(P) %*% D %*% solve(Q) # = M
   [,1] [,2] [,3]
[1,]   2  4  4
[2,]  -6  6 12
[3,]  10 -4 -16
R> det(P)
[1] 1
R> det(Q)
[1] -1
```

### 5.3 Design Philosophy

The `m2r` package was designed with three basic principles in mind:

1. make Macaulay2 as R-user friendly as possible

2. be as flexible with Macaulay2 syntax and data structures possible

3. minimize computational overhead
We work towards these goals with a functional approach by including new data structures, a robust Macaulay2 parser, lazy parsing, and reference functions. In this section we describe these in at a moderate level of detail; for more information, we direct the reader to the GitHub page at https://github.com/coneill-math/m2r.

### 5.3.1 m2r Data Structures

One of the challenges of developing a computer algebra system for R is that R has no infrastructure to handle algebraic objects. mpoly alleviates this for polynomials, but not for rings, ideals, or any other object in computational algebraic geometry.

For complete details on the m2r parser, see [58]; here we only describe the basic functionality. Most objects parsed from Macaulay2 back into R are objects whose last class type is "m2" and whose other class types describe the object in decreasing order of specificity. For example:

```r
R> str(R)
Classes 'm2_polynomialring', 'm2' atomic [1:1] NA
 ..- attr(*, "m2_name")= chr "m2rintring00000001"
 ..- attr(*, "m2_meta")=List of 3
 .. ..$ vars :List of 4
 .. .. ..$ : chr "t"
 .. .. ..$ : chr "x"
 .. .. ..$ : chr "y"
 .. .. ..$ : chr "z"
 .. ..$ coefring: chr "QQ"
 .. ..$ order : chr "grevlex"
```

Created in Section 5.2.1 R represents the polynomial ring \( \mathbb{Q}[t,x,y,z] \). As algebraic objects, rings have no natural analogue in R, so m2r needs to provide a data structure to represent them. R is an object of class c("m2_polynomialring", "m2"). The value of the object is
NA, a logical(1) vector; this prevents R users from naively operating on the ring itself. m2r typically represents algebraic objects by parsing them into R as NA with two attributes, a name (m2_name) and a list of metadata (m2_meta). Both have accessor functions:

R> m2_name(R)
[1] "m2rintring00000001"
R> m2_meta(R) %>% str

List of 3
$ vars :List of 4
 ..$ : chr "t"
 ..$ : chr "x"
 ..$ : chr "y"
 ..$ : chr "z"
$ coefring: chr "QQ"
$ order : chr "grevlex"

The m2_name attribute is the Macaulay2 variable binding for the object; it’s the name of the object on the Macaulay2 side. The m2_meta attribute contains other information about the object for easy R referencing.

Almost every object returned by m2r functions behaves this way with one major exception: when the object has a natural analogue in R. For example, both R and Macaulay2 have integers and integer matrices, so it makes sense that when a Macaulay2 integer matrix is parsed back into R, R users can manipulate it just like an ordinary R integer matrix. That is what m2r parses the object into, but m2r makes sure that the object retains the knowledge that it is a Macaulay2 object. For example, the integer matrix $P$ created in the Smith normal form example in Section 5.2.3 is such an object:
This is what allowed us to compute its determinant directly in Section 5.2.3 by calling \( \text{det}(P) \).

5.4 The m2r Cloud

Ultimately, every m2r function that uses Macaulay2 invokes \texttt{m2()}. Every time \texttt{m2()} is called, it checks for a connection to a live Macaulay2 instance. If one is not found, \texttt{start.m2()} is run to initialize the Macaulay2 session. In this section we describe how m2r makes this connection between R and Macaulay2. We begin with sockets, the basic mechanism of connection, and then turn to how these connections support a cloud computing framework that migrates computations off-site, enabling Macaulay2 through m2r for Windows users.
5.4.1 A Socket Connection Between R and Macaulay2

m2r uses sockets as the primary form of communication between concurrent R and Macaulay2 sessions. A socket is a low-level transfer mechanism used for interprocess communication. Sockets are commonly used to send and receive data over the internet, but they can also be used to transfer data between processes running on the same machine. Sockets on a given machine are identified by their port number. To initiate a connection, one endpoint (the server) must open a port for incoming connections, to which the other endpoint (the client) can then connect. Communication through a socket is anonymous; a process need not know the location of the other endpoint when it connects to the socket, sends and receives data through the socket, or closes its connection.

The socket setup has two key advantages. First, it enables a single tethered Macaulay2 session to persist for the duration of the active R session, so any variables or functions the user defines in Macaulay2 remain available for future use. Second, the resulting implementation can be easily extended to run the R and Macaulay2 sessions on different machines, which is the focus of Section 5.4.2.

When start.m2() is called, it attempts to initiate a socket connection between R and Macaulay2 following the sequence of events in Figure 17. Once R successfully binds to the socket opened by Macaulay2, the basic infrastructure is in place for R to send Macaulay2 code as character strings to be evaluated; each such code snippet $L$ is simply relayed to Macaulay2 through the socket. After Macaulay2 evaluates $L$, it constructs and returns a string $S$ containing
(i) any error codes, (ii) the number of lines of output, and (iii) the output; see Figure 18 for an illustration.

(a) \(R\) begins by launching an \(M2\) instance, then waits for an available connection on the specified port.

(b) Once launched, \(M2\) opens a socket on the specified port and waits for a connection to be established.

(c) \(R\) connects to the socket, and \(M2\) pauses while it waits to receive data through the socket.

(d) Upon successfully connecting to the socket, \(R\) returns control to the user until \(m2()\) is called.

Figure 17: The socket connection process.

(a) \(R\) sends a \texttt{Macaulay2} source string \(L\) through the socket, and then waits for a response from \(M2\).

(b) \(M2\) evaluates \(L\), sends its response \(S\) back through the socket, and then resumes listening.

Figure 18: Messages are passed back and forth through the socket.

After \texttt{Macaulay2} issues \(S\), it is relayed through the socket to \(R\), which handles errors and returns the output to the user. When the \(R\) session terminates (or the user calls \texttt{m2.stop()}), the
socket connection is closed by R through sending an empty string through the socket signaling end of file (EOF). Upon receiving an empty string and an EOF signal, Macaulay2 closes the socket connection and exits quietly. These steps cleanly kill the Macaulay2 process spawned by R so that no Macaulay2 processes remain orphaned after the R session is terminated.

It is also worth noting that the script run by the spawned Macaulay2 process does not directly contain any user-supplied code. Instead, a Macaulay2 script that establishes the socket connection with R and conforms to all steps outlined above is run.

5.4.2 Macaulay2 in the Cloud

Cloud computing as a service has come into prominence in recent years through the widespread availability of high speed internet connections and the decreasing cost of hardware and its maintenance at scale, among other things. In a cloud computing model, the users of a software system do not need to download the software which they are using, instead they can simply interact with the software of interest via a web or terminal interface. Users call on the remote machine to perform a calculations, and when the remote computations finish the results are returned to the user.

The core benefit of a cloud computing model for m2r is that users no longer have to install Macaulay2 on their local machines. Installing specialized software can be difficult and time consuming, especially for less computer-savvy users, and this can be an insurmountable barrier to entry to algebraic statistics and algebraic methods in general. This issue is compounded for new users who are not sure if a certain software is the correct solution for their problem and so are unwilling to invest the time. Installing Macaulay2 on a Windows machine is a particularly
challenging task, creating a large barrier to entry for potential Windows users of the package. These are common challenges for specialized mathematical software, and like others before us we concluded that a cloud version of our software was a worthwhile venture \(11; 59\).

Amazon Web Services (AWS, available at [https://aws.amazon.com/](https://aws.amazon.com/)) is a subsidiary of Amazon, Inc. that sells cloud computing solutions. AWS’s flagship product is the Amazon Elastic Compute Cloud (EC2), which provides virtual servers of varying performance specs that can be launched remotely on demand. To help users get up and running with m2r and algebraic statistical computing, we have set up a low-performance EC2 instance dedicated to m2r. We chose to use the introductory tier of this product because it suffices for introducing R users to Macaulay2 and Amazon offers it at no cost. It also provides a proof-of-concept model that can be replicated for a user’s own personal cloud. Instructions for setting up such an instance can be found on m2r’s GitHub page [https://github.com/coneill-math/m2r/inst/server](https://github.com/coneill-math/m2r/inst/server).

A few noteworthy implementation details for remotely running m2r are in order. Each remote instance of Macaulay2 is run within a virtual machine managed by Docker ([https://www.docker.com](https://www.docker.com)), an open source software package that allows for sandboxing of applications inside distinct lightweight virtual software containers. Docker containers provide an additional layer of virtualization that isolates key resources of the host machine. This safeguards the host machine in the sense that nothing executed in a container can affect the host machine. Additionally, containers are optimized to be spun up quickly through efficient usage of host machine resources, significantly decreasing the time necessary to start a new session and allowing m2r to connect to on-demand instances of Macaulay2 in seconds.
While there are many similarities in how m2r connects R to local and remote Macaulay2 instances, there are some important differences as well. Instead of the typical m2r flow where an instance of Macaulay2 is launched on the user’s local machine, the server version allows a user to create on-demand Macaulay2 instances on an active EC2 instance. In addition to running and managing all active Docker containers, the EC2 instance has a Python server script that is used to spawn new Docker instances and dispatch ports to new clients. The connection process for a new R client is diagrammed step-by-step in Figure 19.

(a) The client R session connects to the Python server running on the EC2 instance using a static port. The Python server immediately locates an open and unoccupied port $p$ on the EC2 instance.

(b) The Python server launches a new Docker container provisioned with Macaulay2 and other helpful software. Within this sandboxed container, Macaulay2 is launched and given $p$ as the port number on which to expect an incoming connection.

(c) The Python server sends $p$ to the R client, terminates its connection, and begins listening for connections from the next new R client.

(d) The R client, upon receiving $p$ from the Python server, connects via port $p$ to the Macaulay2 instance running in the new Docker container using the same paradigm used for local Macaulay2 instances.

Figure 19: How R connects to a Macaulay2 session on a remote EC2 host.
The first time \texttt{m2()} is run, \texttt{m2r} will automatically connect to the cloud if no local \texttt{Macaulay2} installation is detected. Note that this will always be the case on a Windows machine, since running a local instance of \texttt{Macaulay2} is not supported. To bypass a local installation and connect to the cloud, use the \texttt{cloud} parameter to \texttt{start_m2()}.

\begin{verbatim}
R> stop_m2()
R> start_m2(cloud = TRUE)
Connecting to M2 in the cloud...
done.
R> m2("1+1")
[1] "2"
\end{verbatim}

If the user has the \texttt{Macaulay2} server script running on their own EC2 instance (or any other cloud service for that matter), the URL can be specified with the \texttt{hostname} parameter to \texttt{start_m2()}. From there, everything will work just as if the user were running a local \texttt{Macaulay2} instance.

5.5 Future Directions

There are several directions of future development that we are excited about, including performance enhancements for the parser, support for arbitrary precision numbers and arithmetic with \texttt{gmp} \cite{gmp}, and modifications to \texttt{mpoly} for broader support for multivariate polynomials in \texttt{R} (e.g. matrices of multivariate polynomials). Perhaps more importantly, \texttt{Macaulay2} has a large number of packages for algebraic statistics that could be of interest to \texttt{R} users and the statistics community more broadly. It will prove beneficial to the algebraic statistics community to connect them to \texttt{R} through \texttt{m2r}. 
APPENDIX

COPYRIGHT FORMS

This appendix contains copies of the two copyright agreements for the papers that are partially reproduced in this manuscript.
Consent to Publish
Lecture Notes in Computer Science

When Author is more than one person the expression “Author” as used in this agreement will apply collectively unless otherwise indicated.

§ 1 Rights Granted

Author hereby grants and assigns to Springer International Publishing AG, Cham (hereinafter called Springer) the exclusive, sole, permanent, world-wide, transferable, sub-licensable and unlimited right to reproduce, publish, distribute, transmit, make available or otherwise communicate to the public, translate, publicly perform, archive, store, lease or lend and sell the Contribution or parts thereof individually or together with other works in any language, in all revisions and versions (including soft cover, book club and collected editions, anthologies, advance printing, reprints or print to order, microfilm editions, audiograms and videograms), in all forms and media of expression including in electronic form (including offline and online use, push or pull technologies, use in databases and networks for display, print and storing on any and all stationary or portable end-user devices, e.g. text readers, audio, video or interactive devices, and for use in multimedia or interactive versions as well as for the display or transmission of the Contribution or parts thereof in data networks or search engines), in whole, in part or in abridged form, in each case as now known or developed in the future, including the right to grant further time-limited or permanent rights. For the avoidance of doubt, Springer has the right to permit others to use individual illustrations and may use the Contribution for advertising purposes.

The copyright of the Contribution will be held in the name of Springer. Springer may take, either in its own name or in that of copyright holder, any necessary steps to protect these rights against infringement by third parties. It will have the copyright notice inserted into all editions of the Contribution according to the provisions of the Universal Copyright Convention (UCC) and dutifully take care of all formalities in this connection in the name of the copyright holder.

§ 2 Regulations for Authors under Special Copyright Law

The parties acknowledge that there may be no basis for claim of copyright in the United States to a Contribution prepared by an officer or employee of the United States government as part of that person’s official duties. If the Contribution was performed under a United States government contract, but Author is not a United States government employee, Springer grants the United States government royalty-free permission to reproduce all or part of the Contribution and to authorize others to do so for United States government purposes.

If the Contribution was prepared or published by or under the direction or control of Her Majesty (i.e., the constitutional monarch of the Commonwealth realm) or any Crown government department, the copyright in the Contribution shall, subject to any agreement with Author, belong to Her Majesty.

If the Contribution was created by an employee of the European Union or the European Atomic Energy Community (EU/Euratom) in the performance of their duties, the regulation 31/EEC, 11/EAE (Staff Regulations) applies, and copyright in the Contribution shall, subject to the Publication Framework Agreement (EC Plug), belong to the European Union or the European Atomic Energy Community.

If Author is an officer or employee of the United States government, of the Crown, or of EU/Euratom, reference will be made to this status on the signature page.

§ 3 Rights Retained by Author

Author retains, in addition to uses permitted by law, the right to communicate the content of the Contribution to other scientists, to share the Contribution with them in manuscript form, to perform or present the Contribution or to use the content for non-commercial internal and educational purposes, provided the Springer publication is mentioned as the
original source of publication in any printed or electronic materials. Author retains the right to republish the Contribution in any collection consisting solely of Author’s own works without subject to ensuring that the publication by Springer is properly credited and that the relevant copyright notice is repeated verbatim. Author may self-archive an author-created version of his/her Contribution on his/her own website and/or the repository of Author’s department or faculty. Author may also deposit this version on his/her funder’s or funder’s designated repository at the funder’s request or as a result of a legal obligation. He/she may not use the publisher’s PDF version, which is posted on SpringerLink and other Springer websites, for the purpose of self-archiving or deposit. Furthermore, Author may only post his/her own version, provided acknowledgment is given to the original source of publication and a link is inserted to the published article on Springer’s website. The link must be accompanied by the following text: “The final publication is available at link.springer.com”.

Prior versions of the Contribution published on non-commercial pre-print servers like ArXiv/CoRR and HAL can remain on these servers and/or can be updated with Author’s accepted version. The final published version (in pdf or html/xml format) cannot be used for this purpose. Acknowledgment needs to be given to the final publication and a link must be inserted to the published Contribution on Springer’s website, accompanied by the text “The final publication is available at link.springer.com”.

Author retains the right to use his/her Contribution for his/her further scientific career by including the final published paper in his/her dissertation or doctoral thesis provided acknowledgment is given to the original source of publication. Author also retains the right to use, without having to pay a fee and without having to inform the publisher, parts of the Contribution (e.g. illustrations) for inclusion in future work, and to publish a substantially revised version (at least 30% new content) elsewhere, provided that the original Springer Contribution is properly cited.

§ 4 Warranties
Author warrants that the Contribution is original except for such excerpts from copyrighted works (including illustrations, tables, animations and text quotations) as may be included with the permission of the copyright holder thereof, in which case(s) Author is required to obtain written permission to the extent necessary and to indicate the precise sources of the excerpts in the manuscript. Author is also requested to store the signed permission forms and to make them available to Springer if required.

Author warrants that he/she is entitled to grant the rights in accordance with Clause 1 “Rights Granted”, that he/she has not assigned such rights to third parties, that the Contribution has not heretofore been published in whole or in part, that the Contribution contains no libelous statements and does not infringe on any copyright, trademark, patent, statutory right or proprietary right of others, including rights obtained through licenses; and that Author will indemnify Springer against any costs, expenses or damages for which Springer may become liable as a result of any breach of this warranty.

§ 5 Delivery of the Work and Publication
Author agrees to deliver to the responsible Volume Editor (for conferences, usually one of the Program Chairs), on a date to be agreed upon, the manuscript created according to the Springer Instructions for Authors. Springer will undertake the reproduction and distribution of the Contribution at its own expense and risk. After submission of the Consent to Publish form Signed by the Corresponding Author, changes of authorship, or in the order of the authors listed, will not be accepted by Springer.

§ 6 Author’s Discount
Author is entitled to purchase for his/her personal use (directly from Springer) the Work or other books published by Springer at a discount of 33 1/3% off the list price as long as there is a contractual arrangement between Author and Springer and subject to applicable book price regulation. Resale of such copies or of free copies is not permitted.

§ 7 Governing Law and Jurisdiction
This agreement shall be governed by, and shall be construed in accordance with, the laws of Switzerland. The courts of Zug, Switzerland shall have the exclusive jurisdiction.

Corresponding Author signs for and accepts responsibility for releasing this material on behalf of any and all Co-authors.

Signature of Corresponding Author: Date:

I’m an employee of the US Government and transfer the rights to the extent transferable (Title 17 §105 U.S.C. applies)

I’m an employee of the Crown and copyright on the Contribution belongs to Her Majesty

I’m an employee of the EU or Euratom and copyright on the Contribution belongs to EU or Euratom
ACM Publishing License and Audio/Video Release

Title of the Work: Computing Tropical Prevarieties in Parallel

Author/Presenter(s): Anders Jensen: Aarhus University; Jeff Sommars: University of Illinois at Chicago; Jan Verschelde: University of Illinois at Chicago

Type of material: Full Paper

Publication and/or Conference Name: International Workshop on Parallel Symbolic Computation Proceedings

1. Glossary

2. Grant of Rights

(a) Owner hereby grants to ACM an exclusive, worldwide, royalty-free, perpetual, irrevocable, transferable and sublicenseable license to publish, reproduce and distribute all or any part of the Work in any and all forms of media, now or hereafter known, including in the above publication and in the ACM Digital Library, and to authorize third parties to do the same.

(b) In connection with software and "Artistic Images and "Auxiliary Materials, Owner grants ACM non-exclusive permission to publish, reproduce and distribute in any and all forms of media, now or hereafter known, including in the above publication and in the ACM Digital Library.

(c) In connection with any "Minor Revision", that is, a derivative work containing less than twenty-five percent (25%) of new substantive material, Owner hereby grants to ACM all rights in the Minor Revision that Owner grants to ACM with respect to the Work, and all terms of this Agreement shall apply to the Minor Revision.

☐ A. Grant of Rights. I grant the rights and agree to the terms described above.

☐ B. Declaration for Government Work. I am an employee of the national government of my country and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only).

Are any of the co-authors, employees or contractors of a National Government?
☐ Yes ☐ No

3. Reserved Rights and Permitted Uses.

(a) All rights and permissions the author has not granted to ACM in Paragraph 2 are reserved to the Owner, including without limitation the ownership of the copyright of the Work and all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM in Paragraph 2(a), Owner shall have the right to do the following:

   (i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.
(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, (3) any repository legally mandated by an agency funding the research on which the Work is based, and (4) any non-commercial repository or aggregation that does not duplicate ACM tables of contents, i.e., whose patterns of links do not substantially duplicate an ACM-copyrighted volume or issue. Non-commercial repositories are here understood as repositories owned by non-profit organizations that do not charge a fee for accessing deposited articles and that do not sell advertising or otherwise profit from serving articles.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work.

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new ACM Consolidated TeX template Version 1.3 and above automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

Please copy and paste the following code snippet into your TeX file between \begin{document} and \maketitle, either after or before CCS codes.

\copyrightyear{2017}\n\acmYear{2017}\n\setcopyright{acmlicensed}\n\acmConference{PASCO 2017}{July 23-24, 2017}{Kaiserslautern, Germany}\n\acmPrice{15.00}\n\acmDOI{.1145/3115936.3115945}\n\acmISBN{978-1-4503-5288-8/17/07}
(b) In connection with any use by the Owner of the Submitted Version (if accepted) or the Accepted Version or a Minor Revision, Owner shall use best efforts to display the ACM citation, along with a statement substantially similar to the following:

"© [Owner] [Year]. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in {Source Publication}, https://doi.org/10.1145/{number}.

5. Audio/Video Recording

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? ☐ Yes ☐ No

6. Auxiliary Material

Do you have any Auxiliary Materials? ☐ Yes ☐ No

7. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

☐ We/I have not used third-party material.
☐ We/I have used third-party materials and have necessary permissions.

8. Artistic Images

If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part IV and be sure to include a notice of copyright with each such image in the paper.

☐ We/I do not have any artistic images.
☐ We/I have any artistic images.

9. Representations, Warranties and Covenants
The undersigned hereby represents, warrants and covenants as follows:

(a) Owner is the sole owner or authorized agent of Owner(s) of the Work;

(b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;

(c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;

(d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;

(e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and

(f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

I agree to the Representations, Warranties and Covenants.

10. Enforcement.

At ACM's expense, ACM shall have the right (but not the obligation) to defend and enforce the rights granted to ACM hereunder, including in connection with any instances of plagiarism brought to the attention of ACM. Owner shall notify ACM in writing as promptly as practicable upon becoming aware that any third party is infringing upon the rights granted to ACM, and shall reasonably cooperate with ACM in its defense or enforcement.

11. Governing Law

This Agreement shall be governed by, and construed in accordance with, the laws of the state of New York applicable to contracts entered into and to be fully performed therein.
CITED LITERATURE

1. 4ti2 Team: 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces. Available at www.4ti2.de, 2015.


10. Bjöck, G. and Saffari, B.: New classes of finite unimodular sequences with uni-

cloud with polynomial homotopy continuation. In Computer Algebra in Scientific
Computing, 17th International Workshop, CASC 2015, eds, V. Gerdt, W. Koepf,
E. Mayr, and E. Vorozhtsov, volume 9301 of Lecture Notes in Computer Science,
pages 87–100, 2015.

12. Bliss, N. and Verschelde, J.: The method of Gauss-Newton to compute power series solu-

University of Illinois at Chicago, Chicago, 2018.

parallel. 2018.


98, 1986.


Combinatorial Algebraic Geometry, 80:207–224, 2017.

algorithms for rational cones and affine monoids. Available at


VITA

Contact Information
Mathematics, Statistics and Computer Science
University of Illinois at Chicago
724 Science and Engineering Offices
851 S Morgan St, Chicago, IL 60607
sommars1@uic.edu
http://homepages.math.uic.edu/~sommars/

Education

University of Illinois at Chicago, Chicago, IL
Ph.D. in Mathematics, Fall 2013–Spring 2018
Advisor: Jan Verschelde

Wheaton College, Wheaton, IL
M.A.T. in Teaching Mathematics, Fall 2011–Spring 2013
Advisor: Laura Barwegen
Thesis: The Effects of the BRIDGE Program on At-Risk Student Achievement

Wheaton College, Wheaton, IL
B.S. in Mathematics, magna cum laude, Fall 2008–Summer 2011
President’s Award recipient

Work Experience

First Trust Portfolios L.P., Wheaton, IL
Software Developer for Research Team, 2014-present

Publications


Accepted


Under Review

Monodromy Solver: sequential and parallel. With Nathan Bliss, Timothy Duff and Anton Leykin.


Computing Tropical Varieties in Macaulay2. With Carlos Amàndola, Kathlèn Kohn, Sara Lamboglia, Diane Maclagan, Ben Smith, Paolo Tripoli and Magdalena Zajaczkowska. arxiv:1710.10651

Talks


Oct 2015. *Polynomial Homotopy Continuation in the Cloud*. Special Session on Algebraic Statistics, 2015 AMS Fall Central Section Meeting. Chicago, IL.


**Funded Research**

1. **Title**: Polynomial Homotopy Continuation in SageMath  
   **PI(s)**: Jeff Sommars  
   **Source**: Institute for Mathematics and its Applications  
   **Amount**: $\approx$12,000  
   **Period**: 10/15/17 – 10/21/17  
   **Note**: Coding sprint at the Institute for Mathematics and its Applications for eight participants

2. **Title**: A Computer Algebra System for R: m2r through the Cloud with EC2  
   **PI(s)**: David Kahle, Chris O’Neill, and Jeff Sommars  
   **Source**: Amazon Web Services  
   **Amount**: $\approx$14,000  
   **Note**: Currently under review

3. **Title**: A Computer Algebra System for R: Macaulay2 and the m2r Package  
   **PI(s)**: David Kahle, Chris O’Neill, and Jeff Sommars  
   **Source**: National Science Foundation: Supported by Grant Number DMS 1321794  
   **Amount**: $1500  
   **Period**: 5/3/17 – 5/7/17

4. **Title**: A Computer Algebra System for R: Macaulay2 and the m2r Package  
   **PI(s)**: David Kahle, Chris O’Neill, and Jeff Sommars  
   **Source**: National Science Foundation: Supported by Grant Number DMS 1321794  
   **Amount**: $1500  
   **Period**: 10/9/16 – 10/13/16
Travel Awards

1. Event: Mittag-Leffler Institute Program on Tropical Geometry.
   Amount: $\approx 1300$ and accommodation
   Period: 1/15/18 – 2/17/18

   Amount: Accommodation
   Period: 8/14/17 – 8/26/17

   Amount: $150$ and accommodation
   Period: 7/27/17 – 7/29/17

4. Event: Joint Mathematics Meetings, Atlanta, GA
   Amount: $950$
   Period: 1/4/17 – 1/4/17

   Amount: $\approx 700, accommodation, and meals$
   Period: 6/12/16 – 6/18/16

6. Event: Macaulay2 Workshop, University of Warwick
   Amount: $750$ and accommodation
   Period: 5/23/16 – 5/26/16

7. Event: Macaulay2 Workshop, University of Utah
   Amount: $450$ and accommodation
   Period: 5/7/16 – 5/10/16

8. Event: Macaulay2 Workshop, Boise State University
   Amount: $\approx 800$ and accommodation
   Period: 5/27/15 – 5/30/15

   Amount: $400$
   Period: 6/16/14 – 6/20/14
**Software**


*PolyGraph.* Compute symmetry of support sets of polynomial systems. Sage.

**Professional Memberships**

American Mathematical Society
Society for Industrial and Applied Mathematics

**Undergraduate Research Supervised**

Philip Hossu: automatic exploitation of symmetry in polynomial system solving