Experimentation at the Frontiers of Reality in Schubert Calculus

Christopher Hillar, Luis García-Puente, Abraham Martín del Campo, James Ruffo, Zach Teitler, Stephen L. Johnson, and Frank Sottile

Abstract. We describe a general framework for large-scale computational experiments in mathematics using computer resources that are available in most mathematics departments. This framework was developed for an experiment that is helping to formulate and test conjectures in the real Schubert calculus. Largely using machines in instructional computer labs during off-hours and University breaks, it consumed in excess of 350 GigaHertz-years of computing in its first six months of operation, solving over 1.1 billion polynomial systems.

Introduction

Mathematical discovery has long been informed by experimentation and computation. Understanding key examples is typically the first step towards formulating theorems and devising proofs. The computer age enables many more potentially intricate examples to be studied than ever before. Sometimes, this leads to a fruitful dialog between theory and experiment. Other times, this work leads serendipitously to new ideas and theorems. Many examples are described in the books [**1, 5**].

We believe there is much greater potential for computer-aided experimentation than what has been achieved. This is particularly true for scientific discovery, using advanced computing to study subtle phenomena and amass evidence for the mathematical facts which will become the theorems of tomorrow. Currently, much computer experimentation is (often appropriately) on a fairly small scale. A notable exception is Odlyzko's study [**28**] (using Cray supercomputers) of the zeroes of Riemann's ζ-function on the critical line $\frac{1}{2}+\mathbb{R}\sqrt{-1}$, which led to a rich data set that has stimulated much intriguing mathematics [**8**].

A different large scale use of computers is the Great Internet Mersenne Prime Search (GIMPS) [**22**], which searches for primes of the form 2^p-1 for p a prime, such as $3, 7, 31,$ and 127. Volunteers run software on otherwise idle computers to search for Mersenne primes. This project has found the largest known primes since it started in 1996. Daily, it uses over 60 GigaHertz-years of computation.

This experiment conducted in part on computers provided by NSF grant DMS-0922866.

Research of Sottile supported in part by NSF grant DMS-070105.

Research of Hillar supported in part by an NSF Postdoctoral Fellowship and an NSA Young Investigator grant.

©0000 (copyright holder)

GIMPS is a mathematical analog of big-science physics. We feel there is more scope for such investigations in mathematics. We describe our use of a supercomputer to study a conjecture in the real Schubert calculus, which may serve as a model for research in mathematics based on computational experiments. Rather than Odlyzko's Cray supercomputers, or GIMPS's thousands of volunteers, we use more pedestrian computer resources that are available to many mathematics departments together with modern (and free) software tools such as Perl [**48**], MySQL [**46**], and PHP [**49**], as well as freely available mathematical software such as Singular [**16**], Macaulay 2 [**15**], and Sage [**45**].

This is a methods paper whose purpose is to explain the framework we developed. We do not present mathematical conclusions from this ongoing computational experiment, but instead explain how you, the reader, can take advantage of readily available yet often underutilized computer resources to employ in your mathematical research.

To get an idea of the available resources, in its first six months of data acquisition, this experiment used over 350 GigaHertz-years of computing primarily on 191 computers in instructional labs that are maintained by the Department of Mathematics at Texas A&M University [**18**]. When the labs are not in use, the machines become a cluster computing resource that provides over 500 computational cores for a peak performance of 1.971 Teraflops with 296GB of total memory. This experiment uses a supercomputer moonlighting from its day job of calculus instruction.

The authors of this note include Johnson, who configured the labs as a Beowulf cluster, enabling their use for this computation. Our software was written and maintained by the remaining authors, who include current and former postdocs and students working with Sottile. We are organized into a vertically-integrated team where the senior members work with and mentor the junior members. The overall software design and much of its implementation is due to Hillar.

A key feature of this experiment is its robustness—it can and has recovered from many faults, including emergency system shutdown, database erasure, inexplicable computer malfunction, as well as day-to-day network failures. It is also repeatable, using a pseudorandom number generator with fixed seeds. This repeatability will allow us to rerun a large segment of our calculations on a different supercomputer [**19**] using different mathematical software than the initial run. This will be an unprecedented test of the efficacy of different implementations of our basic mathematical algorithms of Gröbner basis computation and real root counting.

This experiment is part of a long-term study of a striking conjecture in the real Schubert calculus made by Boris Shapiro and Michael Shapiro in 1993. This includes two previous large computational experiments [**31, 40**] (and several smaller ones [**30, 41**]), as well as more traditional work [**10, 11, 20, 38, 39**], including proofs of the Shapiro Conjecture for Grassmannians [**24, 25**]. This story was the subject of a Current Events Bulletin Lecture at the 2009 AMS meeting and a forthcoming article in the AMS Bulletin [**44**].

This experiment is possible only because we may model the geometric problems we study on a computer, efficiently solve them, record the results, and automate this process. We describe some background in Section 1 and the mathematics of the computations in Section 2. In Sections 3–6, we explain the resources (human, hardware, and software) we utilized, the architecture of the experiment, how we ran it on a cluster, and the measures that we took to maintain the quality of our data. We end with some conclusions and remarks.

1. The Shapiro Conjecture and Beyond

Our goal is to describe the design and execution of a large scale computation, which may serve as a model for other experiments in mathematics. While many aspects of our experiment are universal, the details are specific to the questions we are studying. We give some mathematical background to provide context.

Some solutions to a system of real polynomial equations are real and the rest occur in complex conjugate pairs. While the structure of the equations determines the total number of solutions, the distribution between the two types depends subtly on the coefficients. Surprisingly, sometimes there is additional structure which leads to finer information in terms of upper bounds [**3, 21**] or lower bounds [**9, 36**] on the number of real solutions. The Shapiro Conjecture is the extreme situation of having *only* real solutions.

We give an example. Set $\gamma(t) := (6t^2-1, \frac{7}{2}t^3+\frac{3}{2}t, -\frac{1}{2}t^3+\frac{3}{2}t)$, which is a curve $\gamma\colon \mathbb{R} \to \mathbb{R}^3$. We ask for the finitely many lines that meet four tangent lines to γ, which we take to be tangent at the points $\gamma(t)$ for $t = -1, 0, 1$, and some point $s \in (0, 1)$. The first three tangents lie on the quadric Q defined by $x^2 - y^2 + z^2 = 1$. We show this in Figure 1, where $\ell(t)$ is the tangent line at the point $\gamma(t)$.

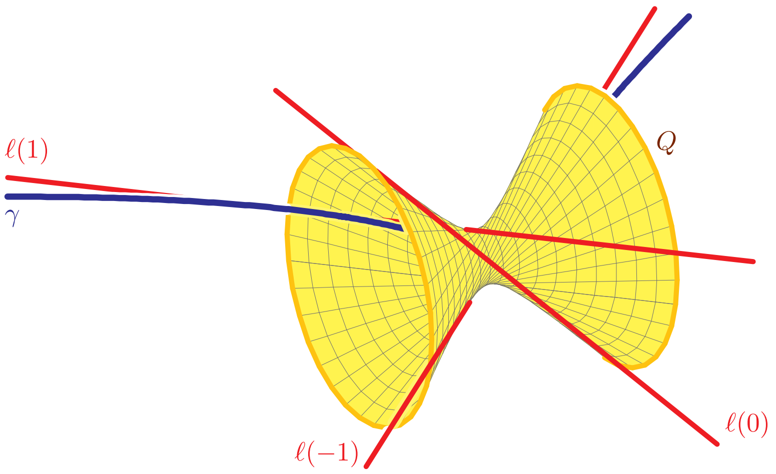

FIGURE 1. Quadric containing three lines tangent to the curve γ.

These first three tangents lie on one ruling of Q and the lines in the other ruling are those meeting them. Lines meeting all four tangents correspond to the (two) *a priori* complex points where the fourth tangent $\ell(s)$ meets the quadric. As we see in Figure 2, for any $s \in (0, 1)$, $\ell(s)$ meets the quadric in two real points, giving two real solutions to this instance of the problem of four lines.

The Schubert calculus [**13, 14**] asks for the linear spaces that have specified positions with respect to other, fixed (flags of) linear spaces. For example, what are the 3-planes in $\mathbb{C}^7$ meeting 12 given 4-planes non-trivially? (There are 462 [**32**].) The specified positions are a *Schubert problem*, for example, the Schubert problem of lines meeting four lines in 3-space. The fixed linear spaces imposing the conditions give an *instance* of the Schubert problem, so that the lines $\ell(-1)$, $\ell(0)$, $\ell(1)$,

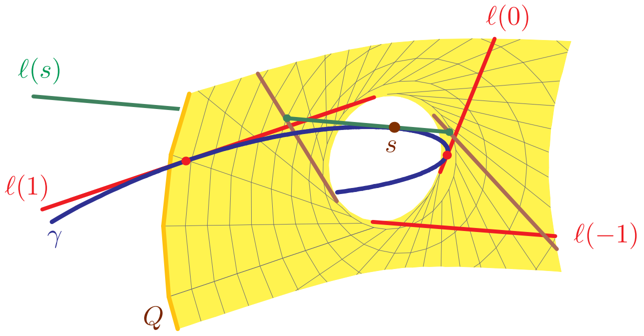

FIGURE 2. Configuration in the throat of the quadric.

and $\ell(s)$ give an instance of the problem of four lines. The number of solutions depends upon the Schubert problem, while the solutions depend upon the instance.

The Shapiro Conjecture begins with a rational normal curve $\gamma\colon \mathbb{R} \to \mathbb{R}^n$, which is any curve projectively equivalent to the moment curve,

$$t \longmapsto (t, t^2, \ldots, t^n)\,.$$

In 1993, Boris Shapiro and Michael Shapiro conjectured that if the fixed linear spaces osculate a rational normal curve, then all solutions to the Schubert problem are real. Initially, the statement seemed too strong to be possibly true. This perception changed dramatically after a few computations [**30, 37**], leading to a systematic study of the conjecture for Grassmannians, both theoretical and experimental [**40**] in which about 40,000 instances were computed of 11 different Schubert problems. Several extremely large instances were also verified by others [**12, 50**].

This early study led to a proof of the Shapiro Conjecture in a limiting sense for Grassmannians [**38**] and a related result in the quantum cohomology of Grassmannians [**39**], which drew others to the area. Eremenko and Gabrielov [**10**] proved it for Grassmannians of codimension 2 subspaces where the Shapiro Conjecture becomes the statement that a univariate rational function with only real critical points is (equivalent to) a quotient of real polynomials.

Later, Mukhin, Tarasov, and Varchenko [**24**] used ideas from integrable systems to prove the Shapiro Conjecture for Grassmannians. They later gave a second proof [**26**] that revealed deep connections between geometry and representation theory. This story was the subject of a Current Events Bulletin Lecture at the January 2009 AMS meeting and a forthcoming article in the AMS Bulletin [**44**].

1.1. Beyond the Grassmannian. The Shapiro Conjecture makes sense for any flag manifold (compact rational homogeneous space). Early calculations [**41**] supported it for orthogonal Grassmannians but found counterexamples for general SL_n-flag manifolds and Lagrangian Grassmannians. Calculations suggested modifications in these last two cases [**43**] and limiting versions were proved [**41**], and the conjecture for the orthogonal Grassmannian wasjust proven by Purbhoo [**29**].

The modification for SL_n-flag manifolds, the *Monotone Conjecture*, was refined and tested in a computational experiment involving Ruffo and Sottile [**31**]. That ran on computers at the University of Massachusetts, the Mathematical Sciences Research Institute, and Texas A&M University, using 15.76 GigaHertz-years of computing to study over 520 million instances of 1126 different Schubert problems on 29 flag manifolds. Over 165 million instances of the Monotone Conjecture were verified, and the investigation discovered many new and interesting phenomena.

For flags consisting of a codimension 2 plane lying on a hyperplane, the Monotone Conjecture is a special case of a statement about real rational functions which Eremenko, et. al [**11**] proved. Their work leads to a new conjecture for Grassmannians. A flag is *secant along an interval I* of a curve if every subspace in the flag is spanned by its intersections with I. The *Secant Conjecture* asserts that if the flags in a Schubert problem on a Grassmannian are *disjoint* in that they are secant along disjoint intervals of a rational normal curve, then every solution is real. It is true for Grassmannians of codimension 2 subspaces, by the result of Eremenko, et. al.

Consider this for the problem of four lines. The hyperboloid in Figure 3 contains three lines that are secant to γ along disjoint intervals. Any line secant along the

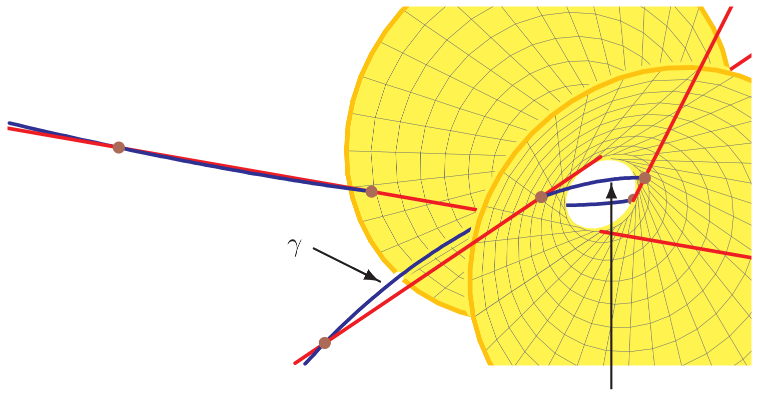

Figure 3. The problem of four secant lines.

indicated arc (which is disjoint from the other three intervals) meets the hyperboloid in two points, giving two real solutions to this instance of the Secant Conjecture.

We are testing the Secant Conjecture for many Schubert problems on small Grassmannians of k-planes in n-space. (See Table 1.) Instances of the Secant

$k \backslash n-k$	2	3	4	5
2	1	5	22	81
3	5	63	94	
4	22			
5	81			

Table 1. Schubert problems studied on $G(k,n)$ as of 20 May 2009.

Conjecture are substantially more difficult to compute than instances of the Shapiro Conjecture. Consequently, this experiment has used much more computing than experiments for the Shapiro and Monotone Conjectures.

2. Solving Schubert problems

Our core mathematical task is the following: Given a Schubert problem on a Grassmannian and secant flags to a rational normal curve, formulate the Schubert problem as a system of equations and determine its number of real solutions.

A Schubert problem is a list $(w_1, \ldots, w_s)$ of conditions to be imposed on k-planes in n-space. A fixed flag instantiating the condition w_i is secant along some m_i points of the rational normal curve γ, so we need $m := m_1 + \cdots + m_s$ points of γ for the given Schubert problem. For the problems we study, $8 \le m \le 52$. Given a set T of m numbers, we use the points $\{\gamma(t) \mid t \in T\}$ to construct secant flags.

Given these secant flags, we formulate the Schubert problem in local coordinates for the Grassmannian (see [**13, 31, 40**] for details), obtaining a set of equations whose common zeroes represent the solutions to the Schubert problem in the local coordinates. We then eliminate all but one variable from the equations, obtaining an *eliminant*. The Shape Lemma [**42**] implies that number of real roots of this eliminant equals the number of real solutions to the original Schubert problem, provided that it is square-free and has degree equal to the expected number of complex solutions, which is easily checked.

This procedure counts the real solutions to a Schubert problem involving secant flags. To compute an instance of the Secant Conjecture, we select m real numbers T and use them to form disjoint secant flags. We also compute cases when the flags are not disjoint. To each set T, we compute five problems involving secant flags. The first is an instance of the Secant Conjecture, but for each of the other four, we randomly alter the assignment of points to get secant flags that are not necessarily disjoint. The *overlap number* measures how far the flags are from disjoint. It is zero if and only if the flags are disjoint.

For each Schubert problem, we perform these steps thousands to millions of times. We repeatedly select subsets T of m numbers from a fixed set of 111 rational numbers, those p/q with $p^2 + q^2 \le 121$. These have small arithmetic height (which affects computation speed). Figure 4 shows these points along $\mathbb{RP}^1$.

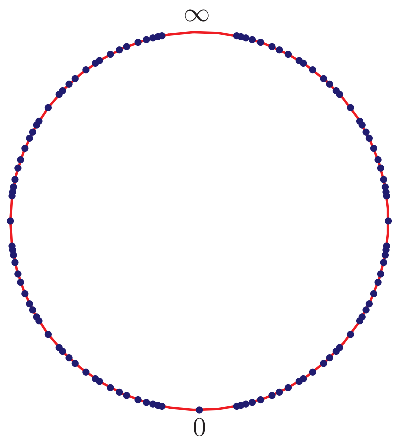

FIGURE 4. 111 points along $\mathbb{RP}^1$.

Table 2 shows the results for a Schubert problem with 16 solutions on $G(3,6)$, the Grassmannian of 3-planes in 6-space. There were $20,000,000$ computed instances of this problem which used 4.473 GigaHertz-years. The rows are labeled with the even integers from 0 to 16 as the number of real solutions has the same parity as the number of complex solutions. The columns are labeled with overlap number, but only the first few and the summary column are shown. The column

$\backslash$	0	1	2	3	4	5	6	$\cdots$	Total
0							20	$\cdots$	7977
2							116	$\cdots$	88578
4				6154	23561	526	3011	$\cdots$	542521
6				25526	63265	2040	9460	$\cdots$	1571582
8				33736	78559	2995	13650	$\cdots$	2834459
10				25953	39252	2540	11179	$\cdots$	3351159
12				35578	44840	3271	14160	$\cdots$	2944091
14				17367	17180	1705	7821	$\cdots$	1602251
16	4568553		182668	583007	468506	36983	83169	$\cdots$	7057382
Total	4568553		182668	727321	735163	50060	142586	$\cdots$	20000000

TABLE 2. Real solutions v.s. overlap number.

with overlap number 0 represents tests of the Secant Conjecture. Since its only entries are in the row for 16 real solutions, the Secant Conjecture was verified in $4,568,553$ instances. The column labeled 1 is empty because flags for this problem cannot have overlap number 1. The most interesting feature is that for overlap number 2, all solutions were real, while for overlap numbers 3, 4, and 5, at least 4 of the 16 solutions were real, and only with overlap number 6 and greater does the Schubert problem have no real solutions. This inner border, which indicates that the reality of the Schubrt problem does not completely fail when there is small overlap, is found on many of the other problems that we investigated and is a new phenomenon that we do not understand.

3. Resources for this experiment

Creating the software and managing this computation is a large project. To accomplish it, we formed a vertically-integrated team of graduate students and postdoctoral fellows under the direction of a faculty member and used modern software tools (Perl, MySQL, PHP) to automate the computation as well as store and visualize data. This software runs on many different computers, but primarily on a supercomputing cluster whose day job is calculus instruction.

3.1. People. The authors of this note include Johnson, who created and maintains the supercomputer we use, as well as a research team of current and former graduate students and postdoctoral fellows who have worked with Sottile. Pooling our knowledge in mathematics and in software development, we shared the work of creating and running this project. This structure enabled the senior members to mentor and train the junior members.

We modeled our team structure on the working environment in a laboratory. This led to a division of labor and to other collaborations. For example, García-Puente and Ruffo wrote the mathematical heart of the computation in a Singular library. Hillar, who had experience in the software industry, provided the conceptual framework and contributed most of the Perl code. He worked on some of this with Martín del Campo, who now maintains the PHP webpages we use to monitor the computation. Sottile and Teitler maintain the software and the shell scripts for controlling the computation and ensuring the integrity of the data, and Teitler rewrote the library of our main mathematical routines in Macaulay 2. This project has led to unrelated research collaborations between Hillar and Martín del Campo and between Sottile and Teitler.

Our team includes two more junior members who have not yet contributed code and will soon include an additional postdoc. The web of collaboration and mentoring is designed to help integrate them into future projects.

3.2. Hardware. All mathematics departments have significant, yet deeply underutilized computing resources available in office desktop computers. There are sociological problems that can arise, for example, when your colleague has email problems while his computer is running your software. While these can be overcome, there are often simpler alternatives. Many institutions have some cluster computing resources, and there are regional and national supercomputers available for research use [**27**]. Computers in instructional labs are another resource. With sufficient interest and a modest expenditure, these can be used for research.

The Texas A&M University mathematics department maintains computers for undergraduate instruction. Johnson, the departmental systems administrator, installed job scheduling software enabling their use as a computing cluster outside of teaching hours. The availability of this resource, as much as our mathematical interests, was the catalyst for this computational experiment. It has been the source of 95% of the computing for this experiment, which also used some desktop computers at Texas A&M University and at Sam Houston State University, as well as personal laptops and clusters at the homes of García-Puente and Sottile.

3.3. Software. The computer programming community has developed a vast library of free, open-source software that mathematicians can use for research purposes. The three software tools that we use the most, other than specialized mathematical software, are Perl, MySQL, and PHP. We selected them because of our familiarity with them and their widely available documentation. In addition to excellent manuals [**33, 35, 47**], there are many web pages showing documentation, tutorials, answers to frequently asked questions, and pieces of code.

The distributed nature of our computation, its size, and the amount of data we store, led us to organize the computation around a database to store the results and status of the computation. For this, we chose MySQL, a freely available high-quality database program. The actual database is located on a Texas A&M University mathematics department server and may be accessed from anywhere in the world. In particular, we can (and do) monitor and manage the computation remotely.

Perl is a general-purpose programming language with especially strong facilities for text manipulation and communication with other programs, including MySQL. We use Perl to connect together the mathematical programs that actually perform our calculations (Singular, Maple) with the database.

These data are viewed through web pages, which are dynamically generated using PHP, a programming language designed exactly for this purpose. Our interface for monitoring the experiment is at our project's web page [**34**].

This model—computations on individual computers controlled by a central database—scales well and is very flexible. It can run on a single computer using a local database (e.g. a personal laptop), on a cluster at one's home or department, or on machines at different institutions.

4. Architecture of Computation

We wanted to conduct a large computational experiment using distributed, heterogeneous computer resources and have the computation be largely automated as well as robust, repeatable, and reliable. To accomplish this, we organized it around a database that records all aspects of the computation. In Section 5 we explain how we run this on a cluster and in Section 6 we discuss measures to enhance the quality of our data. Here, we focus on the organization of the computation in our software: The basic mathematical procedures, interaction with the database, dividing this large computation into reasonable-sized pieces, and lastly selecting problems to compute and setting parameters of the computation.

4.1. Implementation. Our computation is split between three subsystems, a controlling Perl script and mathematical computations in Singular and in Maple. We explain these choices and how it all fits together. We chose Perl for its strengths in text manipulation and its interface with MySQL, our database software.

As explained in Section 2, we need to generate a system of polynomials and compute an eliminant, many times. In previous computations, over 97% of computer resources were spent on computing eliminants. We need the computation to be efficient and to run on freely available software. The methods of choice for elimination are algorithms based on Gröbner bases. For this, we tested three Gröbner basis packages (CoCoA [**7**], Macaulay 2 [**15**], and Singular [**16**]) on a suite of representative problems. When we made our choice of elimination software in the Autumn of 2007, Singular was by far the fastest.

Given an eliminant, we need to determine its number of real roots, quickly and reliably. This requires a symbolic algorithm based on Sturm sequences [**2**], and we needed software that we could install on our many different computers. While Maple is proprietary software, it has the fastest and most reliable routine, `realroot`, for counting real roots among the software we tested. Maple was also installed on the computers we planned to use and we trust `realroot` completely, having used it on several billions of previous computations.

The mathematical routines of elimination and real root counting are symbolic (i.e., exact) algorithms. We know of no satisfactory parallel implementations, so we achieve parallelism by running different computations on different CPU cores.

When our software (a Perl script) is run, it queries the database for a Schubert problem to work on and then writes a Singular input file to generate the desired polynomial systems and perform the eliminations. As it writes this file, Perl selects random subsets T of our master list of 111 rational numbers, (re)orders them to make secant flags, and computes (and stores) the overlap number for each polynomial system. After the file is written, Perl calls Singular to run this file to compute the eliminants and write them to a file. Perl then uses that output file to create an input file for Maple, which it calls Maple to run. Maple determines the number of real roots of each eliminant, writing that to a file. Finally, Perl reads Maple's output, pairs the numbers of real roots with the corresponding overlap number, posts these results to the database, and updates the state of the computation.

4.2. Database. A database is just a collection of tables containing data, together with an interface that allows efficient queries. We designed a database to organize this computation. It contains the Schubert problems to be studied, the results (e.g. Table 2), and much else in between. At all times, the database contains a complete snapshot of the calculation. Despite the size of this computation, the database is quite small, about 750 Kilobytes. We briefly explain some of the more important tables in our database and their role in this experiment.

Points contains the master list of 111 numbers used to construct secant flags. It is never altered.

SchubertProblems contains the list of all Schubert problems we intend to study. Section 4.4 explains how we add problems to the database.

Requests keeps track of how much computation is to be done on each Schubert problem and what has been started. The administrators manually update Requests to request more computation for particular problems, and the Perl script updates Requests when beginning a new computation on a Schubert problem.

Results stores the frequency table of real solutions vs. overlap number and the amount of computing for each Schubert problem. The Perl script updates Results after successfully completing a run. This table contains the information that our PHP web pages display.

RunningInstance contains a list of the computations that have started but have yet to be completed. We describe its function in Section 6.4.

4.3. Packets. An important technical aspect of this computation is how we parcel out our computations to individual computers. There are many constraints. Disk space is finite and large files are difficult to handle. Some machines are available only for fixed time periods, and we must efficiently schedule their use. Networks and servers have fixed capacity, so database queries should be kept to a minimum. Additionally, our computations require vastly different resources, with some Schubert problems taking less than 0.033 GigaHertz-seconds per instance while others we studied require in excess of $40,000$ GigaHertz-seconds per instance.

To balance these constraints, we divide the computation of each Schubert problem into units that we call *packets*. Each packet consists of between five and $50,000$ instances (one to $10,000$ choices of the set T), and ideally requires about 1 hour of computation. The packets are processed through one or more Singular/Maple input files, none containing more than 500 polynomial systems. When a computer queries the database for a problem, it is given a single packet to compute.

The database stores the size and composition of the packets (which is set when the problems are loaded into the database), and all information it records on the amount of computation is denominated in these packets.

Packets for computationally-intensive Schubert problems require more than one hour of computing. Schubert problems are sorted by the expected time required for a packet, and this is used in job scheduling to optimize performance. The largest computations are performed on machines with no limit on their availability and the others are parceled out according to the fit between the expected length of computation and the computer's availability.

4.4. Loading Problems. Schubert problems are loaded into the database and the parameters of the computation are set using different software than the main computation. We have code to generate all Schubert problems on a given Grassmannian, determining the number of solutions to each Schubert problem. This uses a Gröbner bases computation in a standard presentation of the cohomology ring, together with the Giambelli formula for Schubert classes [**13**].

A Perl script tests a subset of these problems to determine if the computation is feasible. An administrator selects feasible problems to load into the database with a software tool that runs several instances of each problem and decides, based upon the length of the computation, how to divide the computation into packets. It writes these data into the database and records its work in a log file.

5. Computing on a Beowulf cluster

A *Beowulf cluster* is a simple way to organize computers to work together in which one machine (the server) communicates with the others (its clients), but there is no communication between the clients. This model is optimal for performing many independent computations, for example, when running several computations (e.g. computing Gröbner bases) in parallel. It is a perfect match for our computational needs, and most of our experiment runs on a Beowulf cluster. We describe our cluster, its job scheduling, and how we organized our use of this resource.

5.1. The Calclabs cluster. In Section 3.2 we mentioned our use of instructional computers at Texas A&M University which are collectively called the Calclabs [**18**], as they are primarily used in Calculus classes. The Calclabs consist of 191 computers in five instructional labs and 12 in another lab. Johnson installed the open source batch job scheduler Torque Resource Manager [**17**] on these computers, which are the clients, and on a server. Users log in to the server to submit jobs to a queue from which jobs are given to computers as they becomes available.

Jobs are submitted to the queue with a specified time limit, both for administration and because each computer is available only for a limited time period (typically nights, weekends, and holidays). Jobs exceeding their time limit are terminated. A computer is given the first job (if any) whose time limit does not exceed its availability. As described in Section 4.3, we sort Schubert problems by the expected time to compute a packet to optimize this aspect of the scheduler.

5.2. Scripting and cron. While we monitor the progress of our computation on the Calclabs and sometimes submit jobs to the queue manually, we have largely automated the administration of this computation with the Unix utility cron. Cron executes scheduled tasks and is ideal for performing this administration.

We have set up cron on our account on the server to run a shell script which monitors the queue, submitting jobs when the queue runs down. It does this intelligently, ensuring that the queue contains packets of differing lengths, tailored to the available computing. This runs once per hour to keep the queue well-stocked. Other administrative tasks (rotating logs, deleting old temporary files and archiving the database) are performed once each day.

Since the scheduler submits one job to each machine, but our software runs on a single core, the jobs are themselves shell scripts that run one copy of our software for each CPU core on the given machine.

6. Maintaining data quality

An essential requirement in experimental science is that results are reproducible. This is easy to ensure in computational experiments by using deterministic algorithms reliably implemented in software. A second requirement is proper experimental design to ensure that a representative sample has been tested. Computational experiments may marry these two requirements by using (pseudo) random number generators with fixed seeds for random sampling, and storing the seeds.

We explain our choices for experimental design and how we ensure the reproducibility of our computation. In principle, this experiment could be rerun, recreating every step in every computation. This repeatability was essential for software development and testing, for checks on the integrity of our data, and it will allow us to rerun a large part of the experiment using different software on a different cluster. With a computation of this complexity, failures of the software and networks are inevitable. We explain how we recover from such failures, both those we anticipate and those that we do not.

6.1. Experimental design. Schubert problems come in a countably infinite family with only a few tens of thousands small enough to model on a computer. We study many of the computable Schubert problems, and for each, we test thousands to millions of instances of the Secant Conjecture.

Previous computations have shown the value of such indiscriminate testing. The seminal example of the Monotone Conjecture (the cover illustration for the issue of Experimental Mathematics in which the paper [**31**] appeared) was tested late in that experiment, and only after the undergraduate member of that team asked why we were omitting it. (Sottile mistakenly thought it would be uninteresting.) Also, extensive initial tests of the Shapiro Conjecture for flag manifolds appeared to affirm its validity (it is in fact false). Later was it realized that, by poor experimental design, only cases of the Monotone Conjecture had been tested, thereby overlooking counterexamples to the Shapiro Conjecture.

We kept these lessons in mind when designing the current experiment. We were indiscriminate in selecting problems, studying all Schubert problems on Grassmannians in 4-, 5-, and 6-dimensional space, as well as on $G(2,7)$ and $G(5,7)$, where $G(k,n)$ is the Grassmannian of k-planes in n-space. We have also studied many computable problems on $G(3,7)$ and will study many on $G(4,7)$, $G(5,7)$, $G(2,8)$, $G(3,8)$, and $G(4,8)$. For these last six Grassmannians, we are computing a random selection of problems. While it is hard to be precise, of the 7286 Schubert problems on $G(4,8)$, we estimate that 3000 could be studied with our software. The rest are too large to compute in a reasonable time or are infeasible.

For a problem involving Schubert conditions $(w_1,\ldots,w_s)$, there are $s!$ ways to order the intervals for secants. Our software randomly reorders the conditions before constructing secant flags, to remove bias from the given ordering. More serious is the question of how uniformly we are selecting from among all secant flags. We do not have a satisfactory answer to this. While one may believe that random subsets of our 111 master numbers (shown in Figure 4) are fairly uniform modulo the action of $PSL(2,\mathbb{R})$, we instead offer experience gained in the previous experiment studying the Monotone Conjecture [**31**]. There, the results of a computation (e.g. verifying the conjecture and an inner border as in Table 2) did not appear to depend upon how we selected subsets of a master set of numbers. The selections included such schemes as all subsets of the numbers $\{1,\ldots,10\}$, or random subsets of the first 20 prime numbers, or random subsets of all rational numbers p/q where (p,q) are the integer points closest to $\bigl(101\cos(\frac{\pi}{40}n), 101\sin(\frac{\pi}{40}n)\bigr)$, for $n=1,\ldots,40$. (This last scheme is likely nearly uniform.)

6.2. Reproducibility and seeds. Random choices are made with the help of a pseudorandom number generator. Its output depends deterministically, but to all appearances unpredictably, on a state variable called a *seed*, which is deterministically updated after each call. Thus two sequences of calls to a pseudorandom number generator beginning with different seeds give unrelated sequences of integers, but if the seeds are the same, the sequences are identical.

We take advantage of this by generating an initial seed for each Schubert problem before computing its first packet. This initial seed is determined by the current state of the computer. When packet n is begun, the seed is set to this initial seed and n calls are made to the pseudorandom number generator to set the seed for that packet. In this way, the computation of a given Schubert problem is completely determined given this initial seed, which is stored in our database.

6.3. Independent Checks. This exact reproducibility of results in a computational experiment is much stronger than the notion of reproducibility for statistical results, and is a feature that we exploit. We use it for software development to test upgrades and to ensure that the software runs properly on different machines. For this, we simply copy some problems and their initial seeds to an empty database, run all requested computations, and then compare the new results with the old results. (They have always agreed.)

We have a software tool to automate this process and now use it on individual laptops to rerun the computation for some Schubert problems to validate the data for these problems from the initial run. We understand that GIMPS also uses such double-checking for validation.

More interesting, and we believe unprecedented in mathematical experimentation, we are starting to use this feature to rerun a large segment of the calculation on a different cluster [**19**] running a different version of Linux and different hardware and also using different software for our basic mathematical routines. We use Macaulay 2 for elimination in place of Singular and SARAG [**6**] in place of Maple for counting real roots of univariate polynomials. Besides providing an independent check on the data we generate, this will also give a direct comparison of the efficacy of different implementations of these basic mathematical routines.

6.4. Robustness. As with any complicated task, we cannot avoid the unexpected (the unknown unknowns), and have designed our software to recover from the many different failures that inevitably occur. For this, we have several interlocking systems to prevent corrupted calculations from being entered in the database. We also rerun corrupted packets, and even rerun all or part of a Schubert problem whose data appear suspect.

First, our software has checks to ensure that tasks (connecting with the database, calling client programs, and reading/writing data in files) are successfully executed, and which terminate its running when something untoward is detected. Other errors, such a network errors or unmounted file systems, cause noisier failures, which are captured by our log files for possible diagnosis. There are even less graceful failures in a computation, such as power outage, termination of jobs by the job server, or simple human error. All of these abort the computation of a packet and therefore lead to packets whose computation has started, but whose results have not been submitted to the database.

The table RunningInstance in our database keeps track of packets whose computation has started but not finished, together with the expected completion time. Packets that have been terminated in any way are recognized by having an expected completion time that has long passed. When our software queries the database for a problem to work on, it first checks for any such overdue packets. If one is found, it deletes that record from RunningInstance and creates a new record corresponding to this new computation. Otherwise it finds a fresh packet to compute, creating a record in RunningInstance. Upon successful completion, these records are deleted. One possible graceful failure is for a computation to end, but discover that its record in RunningInstance has been deleted and superseded by another—preventing a second submission of the same data to the database.

While this method works for the few to hundreds (out of thousands) of packets each day that fail before successful completion, sometimes our data becomes corrupted, or possibly corrupted. We also have a software tool that finds the most recent database backup where that Schubert problem is uncorrupted and restores that Schubert problem to this previous state. Thus we simply recompute all or part of the computations for that Schubert problem.

7. Conclusions and future work

We plan to continue this work of mathematical discovery through advanced computing. At the conclusion of this experiment in December 2009, we will write a full paper describing its mathematical results. As of May 2009, the Secant Conjecture was verified in each of the over 250 million instances we checked. In 2010 we plan to start a related experiment, testing a common generalization of the Monotone and Secant Conjectures. This will last about one year. While there is much more to be discovered studying these variants of the Shapiro Conjecture, we plan a long-term, multifaceted, and systematic study of Galois groups of Schubert problems, building on the work in [**4, 23**].

We mention one side benefit from this computation. In March 2009, after sharing timing data from a benchmark computation with Mike Stillman, a developer of Macaulay 2, he rewrote some code that improved its running by several orders of magnitude.

We have described how and why we set up, organized, and are running a very large computational experiment to study a conjecture in pure mathematics, and how it is possible to harness underused yet widely available computing resources for mathematical research. We believe this model—a team-based approach to designing and monitoring a large computational experiment—can fruitfully be replicated in other settings and we encourage you to try it.

References

[1] David H. Bailey, Jonathan M. Borwein, Neil J. Calkin, Roland Girgensohn, D. Russell Luke, and Victor H. Moll, *Experimental mathematics in action*, A K Peters Ltd., Wellesley, MA, 2007.

[2] Saugata Basu, Richard Pollack, and Marie-Françoise Roy, *Algorithms in real algebraic geometry*, Algorithms and Computation in Mathematics, vol. 10, Springer-Verlag, Berlin, 2003.

[3] Daniel J. Bates, Frédéric Bihan, and Frank Sottile, *Bounds on the number of real solutions to polynomial equations*, Int. Math. Res. Not. IMRN (2007), no. 23, Art. ID rnm114, 7.

[4] Sara Billey and Ravi Vakil, *Intersections of Schubert varieties and other permutation array schemes*, Algorithms in algebraic geometry, IMA Vol. Math. Appl., vol. 146, Springer, New York, 2008, pp. 21–54.

[5] Jonathan Borwein and David Bailey, *Mathematics by experiment*, second ed., A K Peters Ltd., Wellesley, MA, 2008.

[6] Fabrizio Caruso, *The SARAG library: some algorithms in real algebraic geometry*, Mathematical software—ICMS 2006, Lecture Notes in Comput. Sci., vol. 4151, Springer, Berlin, 2006, pp. 122–131.

[7] CoCoATeam, CoCoA*: a system for doing Computations in Commutative Algebra*, Available at `cocoa.dima.unige.it`.

[8] Persi Diaconis, *Patterns in eigenvalues: the 70th Josiah Willard Gibbs lecture*, Bull. Amer. Math. Soc. (N.S.) **40** (2003), no. 2, 155–178 (electronic).

[9] A. Eremenko and A. Gabrielov, *Degrees of real Wronski maps*, Discrete Comput. Geom. **28** (2002), no. 3, 331–347.

[10] ———, *Rational functions with real critical points and the B. and M. Shapiro conjecture in real enumerative geometry*, Ann. of Math. (2) **155** (2002), no. 1, 105–129.

[11] A. Eremenko, A. Gabrielov, M. Shapiro, and A. Vainshtein, *Rational functions and real Schubert calculus*, Proc. Amer. Math. Soc. **134** (2006), no. 4, 949–957 (electronic).

[12] J.-C. Faugère, Fabrice Rouillier, and Paul Zimmermann, *Private communication*, 1999.

[13] William Fulton, *Young tableaux*, London Mathematical Society Student Texts, vol. 35, Cambridge University Press, Cambridge, 1997.

[14] William Fulton and Piotr Pragacz, *Schubert varieties and degeneracy loci*, Lecture Notes in Mathematics, vol. 1689, Springer-Verlag, Berlin, 1998.

[15] Daniel R. Grayson and Michael E. Stillman, *Macaulay 2, a software system for research in algebraic geometry*, Available at `www.math.uiuc.edu/Macaulay2/`.

[16] G.-M. Greuel, G. Pfister, and H. Schönemann, SINGULAR *3.0.4 — A computer algebra system for polynomial computations*, 2007, `www.singular.uni-kl.de`.

[17] Cluster Resources Inc., *Torque Resource Manager*, `www.clusterresources.com/pages/products/torque-resource-manager.php`.

[18] Steve Johnson, `calclab.math.tamu.edu/grid/index.xhtml`.

[19] ———, `brazos.tamu.edu`.

[20] Viatcheslav Kharlamov and Frank Sottile, *Maximally inflected real rational curves*, Mosc. Math. J. **3** (2003), no. 3, 947–987, 1199–1200.

[21] A.G. Khovanskii, *Fewnomials*, Trans. of Math. Monographs, 88, AMS, 1991.

[22] S. Kurowski and G. Woltman, *Great Internet Mersenne Prime Search*, 1996, `www.mersenne.org`.

[23] A. Leykin and F. Sottile, *Galois groups of Schubert problems via homotopy computation*, Math. Comp. **78** (2009), 1749–1765.

[24] E. Mukhin, V. Tarasov, and A. Varchenko, *The B. and M. Shapiro conjecture in real algebraic geometry and the Bethe ansatz*, Annals of Math. **170** (2009), No. 2, 863–861.

[25] ———, *Schubert calculus and representations of the general linear group*, J. Amer. Math. Soc. **22** (2009), no. 4, 909–940.

[26] ———, *On reality property of Wronski maps*, 2007, Confluentes Mathematici, to appear.

[27] National Science Foundation, *TeraGrid*, `www.teragrid.org`.

[28] A. M. Odlyzko, *On the distribution of spacings between zeros of the zeta function*, Math. Comp. **48** (1987), no. 177, 273–308.

[29] K. Purbhoo, *Reality and transversality for Schubert calculus in $OG(n, 2n+1)$*, `arXiv:0911.2039`.

[30] J. Rosenthal and F. Sottile, *Some remarks on real and complex output feedback*, Systems Control Lett. **33** (1998), no. 2, 73–80.

[31] J. Ruffo, Y. Sivan, E. Soprunova, and F. Sottile, *Experimentation and conjectures in the real Schubert calculus for flag manifolds*, Experiment. Math. **15** (2006), no. 2, 199–221.

[32] H. Schubert, *Anzahl-Bestimmungen für lineare Räume beliebiger Dimension*, Acta. Math. **8** (1886), 97–118.

[33] Randal Schwartz, Tom Phoenix, and brian d foy, *Learning Perl*, O'Reilly, 2008.

[34] Secant Team, *Frontiers of Reality in Schubert Calculus*, `www.math.tamu.edu/~secant`.

[35] Tahaghoghi Seyed and Hugh Williams, *Learning MySQL*, O'Reilly, 2007.

[36] E. Soprunova and F. Sottile, *Lower bounds for real solutions to sparse polynomial systems*, Adv. Math. **204** (2006), no. 1, 116–151.

[37] Frank Sottile, *Enumerative geometry for real varieties*, Algebraic geometry—Santa Cruz 1995, Proc. Sympos. Pure Math., vol. 62, Amer. Math. Soc., Providence, RI, 1997, pp. 435–447.

[38] ———, *The special Schubert calculus is real*, Electron. Res. Announc. Amer. Math. Soc. **5** (1999), 35–39 (electronic).

[39] ———, *Real rational curves in Grassmannians*, J. Amer. Math. Soc. **13** (2000), no. 2, 333–341.

[40] ———, *Real Schubert calculus: polynomial systems and a conjecture of Shapiro and Shapiro*, Experiment. Math. **9** (2000), no. 2, 161–182.

[41] ———, *Some real and unreal enumerative geometry for flag manifolds*, Michigan Math. J. **48** (2000), 573–592, Dedicated to William Fulton on the occasion of his 60th birthday.

[42] ———, *From enumerative geometry to solving systems of polynomials equations*, Computations in algebraic geometry with Macaulay 2, Algorithms Comput. Math., vol. 8, Springer, Berlin, 2002, pp. 101–129.

[43] ———, *Enumerative real algebraic geometry*, Algorithmic and quantitative real algebraic geometry (Piscataway, NJ, 2001), DIMACS Ser. Discrete Math. Theoret. Comput. Sci., vol. 60, Amer. Math. Soc., Providence, RI, 2003, pp. 139–179.

[44] ———, *Frontiers of reality in Schubert calculus*, AMS Bulletin, to appear, January, 2010.

[45] William Stein, *Sage: Open Source Mathematical Software*, The Sage Group, 2008, `www.sagemath.org`.

[46] Sun Microsystems, Inc., *MySQL*, `www.mysql.com`.

[47] Kevin Tatroe, Rasmus Lerdorf, and Peter MacIntyre, *Programming PHP*, O'Reilly, 2006.

[48] The Perl Foundation, *Perl*, `www.perl.org`.

[49] The PHP Group, *PHP: Hypertext Processor*, `www.php.org`.

[50] Jan Verschelde, *Numerical evidence for a conjecture in real algebraic geometry*, Experiment. Math. **9** (2000), no. 2, 183–196.

Christopher Hillar, The Mathematical Sciences Research Institute, 17 Gauss Way, Berkeley, CA 94720-5070
E-mail address: `chillar@msri.org`

Luis García-Puente, Department of Mathematics and Statistics, Sam Houston State University, Huntsville, TX 77341
E-mail address: `lgarcia@shsu.edu`
URL: `www.shsu.edu/~ldg005`

Abraham Martín del Campo, Department of Mathematics, Texas A&M University, College Station, TX 77843
E-mail address: `asanchez@math.tamu.edu`
URL: `http://www.math.tamu.edu/~asanchez`

James Ruffo, Department of Mathematics, Computer Science, & Statistics, State University of New York, College at Oneonta, Oneonta, NY 13820
E-mail address: `ruffojv@oneonta.edu`
URL: `http://employees.oneonta.edu/ruffojv/`

Zach Teitler, Department of Mathematics, Texas A&M University, College Station, TX 77843
E-mail address: `zteitler@math.tamu.edu`
URL: `http://www.math.tamu.edu/~zteitler`

Stephen L. Johnson, Department of Mathematics, Texas A&M University, College Station, TX 77843
E-mail address: `steve@math.tamu.edu`
URL: `http://www.math.tamu.edu/~steve.johnson`

Frank Sottile, Department of Mathematics, Texas A&M University, College Station, TX 77843
E-mail address: `sottile@math.tamu.edu`
URL: `http://www.math.tamu.edu/~sottile`