Introduction

The terms "Algebra" and "Algorithm" both originate from the same person. Muhammad ibn Musa al Khwarizmi was a 9th century Persian mathematician, astronomer and geographer. The Latin translation of his books introduced the Hindu-Arabic decimal system to the western world. His book "The Compendious Book on Calculation by Completion and Balancing" also presented the first general solution for quadratic equations via the technique of "completing the square". More than that, this book introduced the notion of solving *general* as opposed to specific equations by a sequence of manipulations such as subtracting or adding equal amounts. Al Khwarizmi called the latter operation *al-jabr* ("restoration" or "completion"), and this term gave rise to the word *Algebra*. The word *Algorithm* is derived from the Latin form of Al Khwarizmi's name.[1]

[1] See *The equation that couldn't be solved* by Mario Livio for much of this history.

Figure 1: Muhammad ibn Musa al-Khwarizmi (from a 1983 Soviet Union stamp commemorating his 1200 birthday).

However, the solution of equations of degree *larger than two* took much longer time. Over the years, a great many ingenious people devoted significant effort to solving special cases of such equations. In the 14th century, the Italian mathematician Maestro Dardi of Pisa gave a classification of 198 types of cubic (i.e., degree 3) and quartic (i.e., degree 4) equations, but could not find a general solution for all such examples. Indeed, in the 16th century, Italian mathematicians would often hold "Mathematical Duels" in which opposing mathematicians would present to each other equations to solve. These public competitions attracted many spectators, were the subject of bets, and winning such duels was often a condition for obtaining appointments or tenure at universities. It is in the context of these competitions, and through a story of intrigue, controversy and broken vows, that the general formula for cubic and quartic equations was finally discovered, and later published in the 1545 book of Cardano.

However, the solution for *quintic* formulas took another 250 years. Many great mathematicians including Descartes, Leibnitz, Lagrange,

Euler, and Gauss worked on the problem of solving equations of degree five and higher, finding solutions for special cases but without discovering a general formula. It took until the turn of the 19th century and the works of Ruffini, Galois and Abel to discover that in fact such a general formula for solving degree 5 or higher equations via combinations of the basic arithmetic formulas and taking roots does *not* exist. More than that, these works gave rise to a *precise characterization* of which equations are solvable and thus led to the birth of *group theory*.

Today, solving an equation such as $x^{17} = 1$ (which amounts to constructing a 17-gon using a compass and straightedge- one of the achievements Gauss was most proud of) can be done in a few lines of routine calculations. Indeed, this is a story that repeats itself often in science: we move from special cases to a general theory, and in the process transform what once required creative genius into mere calculations. Thus often the sign of scientific success is when we eliminate the need for creativity and make boring what was once exciting.

Let us fast-forward to present days, where the *design of algorithms* is another exciting field that requires a significant amount of creativity. The Algorithms textbook of Cormen et al. has 35 chapters, 156 sections, and 1312 pages, dwarfing even Dardi's tome on the 198 types of cubic and quartic equations. The crux seems to be the nature of *efficient* computation. While there are some exceptions, typically when we ask whether a problem can be solved *at all* the answer is much simpler, and does not seem to require such a plethora of techniques as is the case when we ask whether the problem can be solved *efficiently*. Is this state of affairs inherent, or is it just a matter of time until algorithm design will become as boring as solving a single polynomial equation?

We will *not* answer this question in this course. However, it does motivate some of the questions we ask, and the investigations we pursue. In particular, this motivates the study of *general algorithmic frameworks* as opposed to tailor-made algorithms for particular problems.[2] There are several such general frameworks, but we will focus on one example that arises from convex programming: the *Sum of Squares (SOS) Semidefinite Programming Hierarchy*. It has the advantage that on the one hand it is general enough to capture many algorithmic techniques, and on the other hand it is specific enough that (if we are careful) we can avoid the "curse of completeness". That is, we can actually prove impossibility results or lower bounds for this framework without inadvertently resolving questions such as P vs NP. The

[2] There is also a practical motivation for this as well: real-world problems often have their own kinks and features, and will rarely match up exactly to one of the problems in the textbook. A general algorithmic framework can be applied to a wider range of problems, even if they have not been studied before.

hope is that we can understand this framework enough to be able to *classify* which problems it can and can't solve. Moreover, as we will see, through such study we end up investigating issues that are of independent interest, including mathematical questions on geometry, analysis, and probability, as well as questions about modeling *beliefs* and *knowledge* of computationally bounded observers.

The Sum of Squares Algorithm

Let us now take a step back from the pompous rhetoric and slowly start getting around to the mathematical contents of this course. It will be mostly be focused on the *Sum of Squares (SOS) semidefinite programming hierarchy*. In a sign that perhaps we did not advance so much from the middle ages, the SOS algorithm is also a method for solving polynomial equations, albeit systems of *several* equations in *several* variables. However, it turns out that this is a fairly general formalism. Not only is solving such equations, even in degree two, NP-hard, but in fact one can often reduce directly to this task from problems of interest in a fairly straightforward manner.[3]

[3] In particular, given a 3SAT formula of a form such as $(\overline{x_7} \vee x_{12} \vee x_{29}) \wedge (x_5 \vee x_7 \vee \overline{x_{32}}) \wedge \cdots$, we can easily translate the question of whether it has a satisfying assignment $x \in \{0,1\}^n$ (where n is the number of variables) into the question of whether the equations $x_1^2 - x_1 = 0, \ldots, x_n^2 - x_1 = 0, P(x) - m = 0$ can be solved where m is the number of clause and $P(x)$ is the degree 6 polynomial obtained by summing for every clause j the polynomial C_j such that $C_j(x)$ equals 1 if x satisfies j^{th} clause and $C_j(x) = 0$ otherwise.

We will be interested in solving such equations over the *real numbers*, and typically in settings where (a) the polynomials in questions are low degree, and (b) obtaining an approximate solution is essentially as good as obtaining an exact solution, which helps avoid at least some (if not all) issues of precision and numerical accuracy. Nevertheless, this is still a very challenging setting. In particular, whenever there is more than one equation, or the degree is higher than two, the task of solving polynomial equations becomes *non convex*, and generally speaking, there can be exponentially many local minima for the "energy function" which is obtained by summing up the square violations of the equations. This is problematic since many of the tools we use to solve such equations involve some form of local search, maintaining at each iteration a current solution and looking for directions of improvements. Such methods can and will get "stuck" at such local minima.

When faced with a non-convex problem, one approach that is used in both practice and theory is to *enlarge the search space*.

Geometrically, we hope that by adding additional dimensions, one may find new ways to escape local minima. Algebraically, this often amounts to adding additional variables, with a standard example being the *linearization* technique where we reduce, say, quadratic equations in n variables into a linear equations in n^2 variables by

letting $y_{i,j}$ correspond to $x_i x_j$. If the original system was sufficiently overdetermined, one could hope that we can still solve for y.

The *SOS algorithm* is a systematic way of enlarging the search space, by adding variables in just such a manner. In the example above it adds in the additional constraint that if the matrix $Y = (y_{i,j})$ would be *positive semidefinite*. That is, that it satisfies $w^\top Y w \geq 0$ for every column vector w. (Note that if Y was in fact of the form $Y_{i,j} = x_i x_j$ then $w^\top Y w$ would equal $\langle w, x \rangle^2 \geq 0$.) More generally, the SOS algorithm is parameterized by a number ℓ, known as its *degree*, and for every set of polynomial equations on n variables, yields a semidefinite program[4] on n^ℓ variables that becomes a tighter and tighter approximation of the original equations as ℓ grows. As the problem is NP complete, we don't expect this algorithm to solve polynomial equations efficiently (i.e., with small degree ℓ) in the most general case, but understanding in which cases it does so is the focus of much research efforts and the topic of this course.

[4] A *linear program* is the task of solving a set of linear inequalities (i.e., finding x that satisfies equations of the form $\sum a_i x_i \leq b$). The set of x's satisfying some linear inequalities is known as a *polyhedron* and is convex. A *semidefinite program* is obtained by adding to a linear program a constraint of the form $M(x) \succeq 0$ where M is a symmetric matrix whose every entry is a linear function of x, and $M \succeq 0$ denotes that M is positive semidefinite (i.e. $w^\top M w \geq 0$ for all vectors w). Geometrically, the intersection of a polyhedron with such a constraint is known as a *spectrahedron*.

History

The SOS algorithm has its roots in questions raised in the late 19*th* century by Minkowski and Hilbert of whether any non-negative polynomial can be represented as a sum of squares of other polynomials. Hilbert realized that, except for some special cases (most notably univariate polynomials and quadratic polynomials), the answer is negative and that there are examples—which he showed to exist by non constructive means—of non-negative polynomial that cannot be represented in this way. It was only in the 1960's that Motzkin gave a concrete example of such a polynomial, namely $1 + x^4y^2 + x^2y^4 - 3x^2y^2$. By the arithmetic-mean geometric-mean inequality, $\frac{1+x^4y^2+x^2y^2}{3} \geq (1 \cdot x^4y^2 \cdot x^2y^4)^{1/3}$ and hence this polynomial is always non-negative. However, it is not hard, though a bit tedious, to show that it cannot be expressed as a sum of squares.

In his famous 1900 address, Hilbert asked as his 17th problem whether any polynomial can be represented as a sum of squares of *rational* functions. (For example, Motzkin's polynomial above can be shown to be the sum of squares of four rational functions of denominator and numerator degree at most 6). This was answered positively by Artin in 1927. His approach can be summarized as follows: given a hypothetical polynomial P that cannot be represented in this form, to use the fact that the rational functions are a field to extend the reals into a "pseudo-real" field $\tilde{\mathbb{R}}$ on which there would actually be an element $\tilde{x} \in \tilde{R}$ such that $P(\tilde{x}) < 0$, and then use a "transfer principle"

to show that there is an actual real $x \in \mathbb{R}$ such that $P(x) < 0$.[5] Later in the 60's and 70's, Krivine and Stengle extended this result to show that any unsatisfiable system of polynomial equations can be certified to be unsatisfiable via a *Sum of Squares (SOS) proof* (i.e., by showing that it implies an equation of the form $\sum_{i=1}^{r} p_i^2 = -1$ for some polynomials $p_1, \ldots, p_r$). This result is known as *the Positivstellensatz*.

[5] This description is not meant to be understandable but to make you curious enough to look it up...

In the late 90's / early 2000's, there were two separate efforts on getting quantitative / algorithmic versions of this result. On one hand Grigoriev and Vorobjov [2001] asked the question of *how large* the degree of an SOS proof needs to be, and in particular Grigoriev [2001][6] proved several lower bounds on this degree for some interesting polynomials. On the other hand Parrilo [2000] and Lasserre [2000/01] independently came up with hierarchies of algorithms for polynomial optimization based on the Positivstellensatz using semidefinite programming. (A less general version of this algorithm was also described by Naum Shor [1987] in a 1987 Russian paper, which was cited by Nesterov in 1999.)

[6] :FIXME in a previous version we also had 1999 as the year for Grigoriev's lower bound. why is that? is there a tech report? maybe we should cite that?

It turns that the SOS algorithm generalizes and encapsulates many other convex-programming based algorithmic hierarchies such as those proposed by Lovász and Schrijver [1991] and Sherali and Adams [1990], and other more specific algorithmic techniques such as linear programming and spectral techniques. As mentioned above, the SOS algorithm seems to achieve a "goldilocks" balance of being strong enough to capture interesting techniques but weak enough so we can actually prove lower bounds for it. One of the goals of this course (and line of research) is to also understand what algorithmic techniques can *not* be captured by SOS, particularly in the setting (e.g., noisy low-degree polynomial optimization) where it seems most appropriate for.

Applications of SOS

SOS has applications to: equilibrium analysis of dynamics and control (robotics, flight controls, ...), robust and stochastic optimization, statistics and machine learning, continuous games, software verification, filter design, quantum computation and information, automated theorem proving, packing problems, etc. (For two very different examples, see Figs. 2 and 3.)

Figure 2: SOS was used to analyze the "falling leaf" mode of the U.S. Navy F/A-18 "Hornet", see A. Chakraborty, P. Seiler, and G. J. Balas, Journal of guidance, control, and dynamics, 34(1):73–85, 2011. (Image credit: Wikipedia)

Figure 3: Bachoc and Vallentin used sum-of-squares to give the best known upper bounds for sphere kissing numbers in higher dimensions. See "New upper bounds for kissing numbers from semidefinite programming", C Bachoc, F Vallentin, Journal of the American Mathematical Society 21 (3), 909-924. (Image credit: A. Traffas)

The TCS vs Mathematical Programming view of SOS

The SOS algorithm is intensively studied in several fields, but different communities emphasize different aspect of it. The main characteristics of the Theoretical Computer Science (TCS) viewpoint, as opposed to that of other communities are:

- In the TCS world, we typically think of the number of variables n as large and tending to infinity (as it corresponds to our input size), and the degree ℓ of the SOS algorithm as being relatively small— a constant or logarithmic. In contrast, in the optimization and control world, the number of variables can often be very small (e.g., around ten or so, maybe even smaller) and hence ℓ may be large compared to it.[7]
- Typically in TCS our inputs are discrete and the polynomials are simple, with integer coefficients and constraints such as $x_i^2 = x_i$ that restrict attention to the Boolean cube. Thus we are less concerned with issues of numerical accuracy, boundedness, etc..
- Traditionally people have been concerned with *exact convergence* of the SOS algorithm— when does it yield an exact solution to the optimization problem. This often precludes ℓ from being much smaller than n. In contrast as TCS'ers we would often want to understand *approximate convergence*– when does the algorithm yield an "approximate" solution (in some problem-dependent sense). Since the output of the algorithm in this case is not actually in the form of a solution to the equations, this raises the question of a obtaining *rounding* algorithms, which are procedures to translate the output of the algorithm to an approximate solution.

[7] Since both time and space complexity of the general SOS algorithm scale roughly like n^ℓ, even $\ell = 6$ and $n = 100$ would take something like a petabyte of memory. This may justify the optimization/control view of keeping n small, although if we show that SOS yields a polynomial-time algorithm for a particular problem, then we can hope that we would be able to then optimize further and obtain an algorithm that doesn't require a full-fledged SOS solver. As we will see in this course, this hope has actually materialized in some settings.

SOS as a "cockroach"

In theoretical computer science we typically define a computational problem P and then try to find the best (e.g., most time efficient orbest approximation factor) algorithm A for this problem. One can ask what is the point in restricting attention to a particular algorithmic framework such as SOS, as opposed to simply trying to find the best algorithm for the problem at hand. One answer is that we could hope that if a problem is solved via a general framework, then that solution would generalize better to different variants and cases (e.g., considering average-case variants of a worst-case problem, or measuring "goodness" of the solution in different ways). This is a general phenomenon that occurs time and again many fields, known under

many names including the "bias variance trade-off", the "stability plasticity dilemma", "performance robustness trade-off" and many others. That is, there is an inherent tension between optimally solving a particular question (or optimally adapting to a particular environment) and being robust to changes in the question/environment (e.g., avoiding "over-fitting"). For example, consider the following two species that roamed the earth few hundred million years ago during the Mesozoic era. The dinosaurs were highly complex animals that were well adapted to their environment. In contrast cockroaches have extremely simple reflexes, operating only on very general heuristics such as "run if you feel a brush of air". As one can tell by the scarcity of "dinosaur spray" in stores today, it was the latter species that was more robust to changes in the environment. With that being said, we do hope that the SOS algorithm is at least *approximately optimal* in several interesting settings.

References

Dima Grigoriev. Linear lower bound on degrees of positivstellensatz calculus proofs for the parity. *Theor. Comput. Sci.*, 259(1-2):613–622, 2001.

Dima Grigoriev and Nicolai Vorobjov. Complexity of null-and positivstellensatz proofs. *Ann. Pure Appl. Logic*, 113(1-3):153–160, 2001.

Jean B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM J. Optim.*, 11(3):796–817, 2000/01. ISSN 1052-6234. doi: 10.1137/S1052623400366802. URL http://dx.doi.org/10.1137/S1052623400366802.

László Lovász and Alexander Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1(2): 166–190, 1991.

Pablo A Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, California Institute of Technology, 2000.

Hanif D. Sherali and Warren P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM J. Discrete Math.*, 3(3):411–430, 1990. ISSN 0895-4801. doi: 10.1137/0403036. URL http://dx.doi.org/10.1137/0403036.

N. Z. Shor. An approach to obtaining global extrema in polynomial problems of mathematical programming. *Kibernetika (Kiev)*, (5): 102–106, 136, 1987. ISSN 0023-1274.