Computational Complexity of Solving Real Algebraic Formulae

James Renegar

School of Operations Research and Industrial Engineering, Cornell University
Ithaca, NY 14853-7501, USA

1. Introduction

We briefly survey recent computational complexity results for certain algebraic problems that are relevant to numerical analysis and mathematical programming. Topics include (i) linear programming, (ii) decision methods and quantifier elimination methods for the first order theory of the reals, (iii) solving real algebraic formulae approximately, and (iv) ill-posed problem instances.

2. Linear Programming

In this section we discuss complexity results for the linear programming problem

$$\begin{aligned} &\text{maximize } c^T x \\ &\text{subject to } Ax \geq b \end{aligned} \tag{2.1}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and A is an $m \times n$ matrix.

In the last four years there has been a vast amount of work on "interior point" algorithms, motivated by Karmarkar's algorithm [16]. Unlike the traditional simplex method which moves from vertex to vertex around the feasible region $\{x; Ax \geq b\}$, interior point methods proceed through the interior $\{x; Ax > b\}$ of the feasible region.

Karmarkar's algorithm is a "projective" interior point algorithm, the basic computation for each iteration being a projective transformation. In the last four years another breed of interior point algorithms has received a lot of attention, "path-following" algorithms. These are closer to traditional numerical analysis than are projective algorithms, having Newton's method at their heart. The best upper bounds known for the complexity of linear programming are based on the analysis of particular path-following algorithms.

Following are the simple ideas behind the first path-following algorithm proven to have a polynomial-time bound. Let α_i^T denote the i-th row of the constraint matrix A.

The *center* of the system of linear inequalities $Ax \geq b$ is the point z which maximizes $\prod_i(\alpha_i^T x - b_i)$, viewed as a function restricted to the feasible region. The center exists and is unique if the feasible region is bounded and has non-empty interior, as we assume in what follows. Equivalently, then, the center is

© The Mathematical Society of Japan, 1991

the maximizer of the strictly concave function $f(x) := \sum_i \ln(\alpha_i^T x - b_i)$, viewed as being defined only on the interior of the feasible region.

The center has a natural physical interpretation which arises from the equations $\nabla f(z) = 0$. Consider each hyperplane $\{x; \alpha_i^T x = b_i\}$ as emitting a force which acts on an arbitrary point x not in the hyperplane. The direction of the force is orthogonal to, and away from, the hyperplane, and its magnitude at x equals the reciprocal of the distance from x to the hyperplane, i.e., the force at x is simply $\alpha_i/(\alpha_i T_x - b_i)$. Then the center is the unique equilibrium point in the interior of the feasible region.

Now assume $k^{(0)}$ is a known strict lower bound on the optimal objective value for (2.1). Let $z^{(0)}$ denote the center of the extended system

$$\begin{aligned} Ax &\geq b \\ c^T x &\geq k^{(0)} \end{aligned} \tag{2.2}$$

and assume that $x^{(0)}$ is known to be a feasible "good" approximation to $z^{(0)}$. We know $x^{(0)}$ but not necessarily $z^{(0)}$. We want to move from $x^{(0)}$ towards an optimal solution for (2.1). A natural way to proceed is to create a new system with center $z^{(1)}$ closer to an optimal solution than $z^{(0)}$, and then move from $x^{(0)}$ to a feasible "good" approximation $x^{(1)}$ for $z^{(1)}$.

To create a new system we simply increase $k^{(0)}$. Of course we must be careful that $k^{(0)}$ not be increased above the optimal objective value for (2.1). Hence, it is natural to replace $k^{(0)}$ by $k^{(1)} = \delta c^T x^{(0)} + (1-\delta)k^{(0)}$ where $0 < \delta \leq 1$. This corresponds to bringing the hyperplane $\{x; c^T x = k^{(0)}\}$ for (2.2) towards $x^{(0)}$, thus causing the equilibrium point to move to another point with better objective value.

To compute an approximation $x^{(1)}$ for $z^{(1)}$ we apply one iteration of Newton's method, beginning at $x^{(0)}$, to the equations $\nabla f^{(1)}(x) = 0$ where $f^{(1)}(x) := \ln(c^T x - k^{(1)}) + \sum_i \ln(\alpha_i^T x - b_i)$. (The special structure of $f^{(1)}$ makes the gradient and Hessian very easy to compute).

It is easily proven that if δ is sufficiently small and $x^{(0)}$ is a "good" approximation to $z^{(0)}$ then $x^{(1)}$ obtained in this manner will be a "good" approximation to $z^{(1)}$. However, to establish noteworthy complexity bounds we need to prove something to the effect that δ need not be "too" small. In [23], the author proved that any δ satisfying $0 < \delta \leq 1/13$ is allowable, i.e., proceeding iteratively the algorithm will then generate points $x^{(j)}$, $j = 0, 1, \ldots$, converging to an optimal solution. In practice, much larger δ are acceptable.

The algorithm terminates with standard procedures for computing an exact optimal solution from sufficiently close approximations.

The number "1/13" arises from an analysis of Newton's method for a carefully chosen coordinate system. The analysis can be greatly simplified by relying on work of Smale [32] as was shown by Renegar and Shub [28]. The latter paper presents a unified complexity analysis for several path-following interior point algorithms.

The best complexity bound known for linear programming is due to Vaidya [38]. He established the bound for a path-following algorithm not suggested to be practical, as it relies on fast matrix multiplication. Vaidya's bound is $O((m+n)^{3/2} nL^2 \log(L) \log\log(L))$ bit operations where (very roughly) L is the number of bits required to specify the particular problem instance, i.e., required to specify A, b and c. This bound is an improvement on the earlier record bound of Gonzaga [11] and Vaidya [37] (derived from a modification of the algorithm described above), which in turn was an improvement on the record bound of Karmarkar [16], which in turn was an improvement on the original polynomial-time bound of Khachiyan [17].

The complexity bounds for interior point algorithms are generally obtained by bounding the number of iterations required by an algorithm and multiplying the bound by the amount of work required per average iteration. The best iteration bound known for an interior point method is $O(\sqrt{m+n}\,L)$ and was established by the author in [23]. The recent record complexity bounds arise from clever ways to (theoretically) reduce the amount of work required per average iteration. Much effort has been expended by researchers (including the author) to decrease the iteration bound, but to no avail.

Other seminal papers in the complexity theory of interior point methods include work by Kojima, Mizuno and Yoshise [18] (motivated by Megiddo [19]), Monteiro and Adler [21], and Ye [40]. The amount of recent literature on interior point algorithms is staggering. Relevant surveys have been written by Gonzaga [12], Goldfarb and Todd [10], and Megiddo [20].

All polynomial time algorithms for linear programming require polynomial time in the Turing machine sense, i.e., the number of bit operations is bounded by a polynomial function in the bit length L of the input. The number of arithmetic operations (over the rationals Q) for all of the algorithms tends to infinity as L does, even when m and n are fixed. (By contrast, the number of arithmetic operations required by the simplex method can be bounded above by a function of m and n alone).

A major open question in the complexity theory of linear programming is whether or not there exist polynomial-time real number machine algorithms in the sense of Blum, Shub and Smale [2], i.e., is "uniform," accepts arbitrary real number coefficients as inputs, and has arithmetic operation count bounded by a polynomial in m and n. Tardos [35] has made some progress on this question by devising an algorithm with arithmetic operation bound independent of the coordinates of the data vectors b and c, but dependent on the coefficients of A which she assumes to be integers.

"Experts" are divided in their opinions as to the answer of the question. If the answer was affirmative then most likely there would be practically important linear programming algorithms yet to be discovered. If the answer was negative then the complexity heirarchy for real number machines would be very different from that for Turing machines.

3. Decision Methods and Quantifier Elimination Methods

Now we move to a very general setting which includes many problems from numerical analysis and mathematical programming, e.g., eigenvalue problems, non-linear programming with multi-variate polynomial objective and constraint functions, sensitivity analysis problems, etc. All of these problems arise in the *classical* (by computational complexity standards) setting of the decision problem for the first order theory of the reals. We begin with a quick introduction for readers unfamiliar with this setting.

A *sentence* is an expression composed from certain ingredients. Letting $\mathbb{R}$ denote a real-closed field, the following is an example of a sentence:

$$(\exists x_1 \in \mathbb{R}^{n_1})(\forall x_2 \in \mathbb{R}^{n_2}) \left[(g_1(x_1,x_2) > 0) \bigvee (g_2(x_1,x_2) = 0)\right] \bigwedge (g_3(x_1,x_2) \neq 0). \tag{3.1}$$

The ingredients are: vectors of variables (x_1 and x_2); the quantifiers $\exists$ and $\forall$; atomic predicates (e.g. $g_1(x_1,x_2) > 0$) which are polynomial inequalities ($>, \geq, =, \neq, <,$); and a Boolean function holding the atomic predicates ($[B_1 \vee B_2] \wedge B_3$).

A sentence asserts something. The above sentence asserts that there exists $x_1 \in \mathbb{R}^{n_1}$ such that for all $x_2 \in \mathbb{R}^{n_2}$, (i) either $g_1(x_1,x_2) > 0$ or $g_2(x_1,x_2) = 0$, and (ii) $g_3(x_1,x_2) \neq 0$. Depending on the specific coefficients of the atomic predicate polynomials this assertion is true or it is false.

The set of all true sentences constitutes the first order theory of the reals. A *decision method* for the first order theory of the reals is an algorithm which, given any sentence, correctly determines if the sentence is true. Decision methods for the reals were first proven to exist by Tarski [36] who constructed one.

A sentence is a special case of a more general expression, called a *formula.* Here is an example of a formula:

$$(\exists x_1 \in \mathbb{R}^{n_1})(\forall x_2 \in \mathbb{R}^{n_2}) \left[(g_1(z,x_1,x_2) > 0) \bigvee (g_2(z,x_1,x_2) = 0)\right] \bigwedge (g_3(z,x_1,x_2) \neq 0). \tag{3.2}$$

A formula has one thing that a sentence does not, namely, a vector $z \in \mathbb{R}^{n_0}$ of *free variables.* When specific values are substituted for the free variables, the formula becomes a sentence.

A vector $\bar{z} \in \mathbb{R}^{n_0}$ is a *solution* for the formula if the sentence obtained by substituting $\bar{z}$ is true.

Two formulae are *equivalent* if they have the same solutions.

A *quantifier elimination method* is an algorithm which, given any formula, computes an equivalent quantifier-free formula, i.e., for the above formula $(\exists x_1 \in \mathbb{R}^{n_1})(\forall x_2 \in \mathbb{R}^{n_2})P(z,x_1,x_2)$ such a method would compute an equivalent formula $Q(z)$ containing no quantified variables.

When a quantifier elimination method is applied to a sentence, it becomes a decision method. Thus, a quantifier elimination method is in some sense more general than a decision method.

Tarski [36] actually constructed a quantifier elimination method.

Many problems in numerical analysis and mathematical programming can be cast as the problem of computing a solution for a particular formula. The reader will easily verify that this can be done for the problems mentioned earlier. It can be done for many other problems as well. Of course determining if a solution for a formula exists can be done with a decision method. In the next section we discuss the complexity of approximating solutions for formulae.

Both (3.1) and (3.2) are said to be in prenex form, i.e., all quantifiers occur in front. More generally, a formula can be constructed from other formulae just as (3.1) was constructed from the atomic predicates.

We now present a brief survey of some complexity highlights for quantifier elimination methods, considering only formulae in prenex form. General bounds follow inductively. (If a formula is constructed from other formulae, first apply quantifier elimination to the innermost formulae, then to the innermost formulae of the resulting formula, etc.)

We consider the general formula

$$(Q_1 x_1 \in \mathbb{R}^{n_1}) \ldots (Q_\omega x_\omega \in \mathbb{R}^{n_\omega}) P(z, x_1, \ldots, x_\omega), \tag{3.3}$$

where $Q_1, \ldots, Q_\omega$ are quantifiers, assumed without loss of generality to alternate, i.e., Q_i is not the same as Q_{i+1}. Let m denote the number of distinct polynomials occurring among the atomic predicates and let $d \geq 2$ be an upper bound on their degrees.

In the case of Turing machine computations where all polynomial coefficients are restricted to be integers, we let L denote the maximal bit length of the coefficients. In this context we refer to the number of "bit operations" required by a quantifier elimination method. In the general and idealized case that the coefficients are not integers we rely on the computational model of Blum, Shub and Smale [2], and refer to "arithmetic operations", these essentially being field operations, including comparisons.

The sequential bit operation bounds that have appeared in the literature are all basically of the form

$$(md)^E \left[L^{O(1)} + \text{Cost}\right] \tag{3.4}$$

where E is some exponent and "Cost" is the worst-case cost of evaluating the Boolean function holding the atomic predicates, i.e., worst-case over 0–1 vectors.

The first reasonable upper bound for a quantifier elimination method was proven by Collins [5]. He obtained $E = 2^{O(n)}$ where $n := n_0 + \ldots + n_\omega$. Collins' bound is thus "doubly exponential" in the number of variables. His method requires the formula coefficients to be integers, the number of arithmetic operations (not just bit operations) growing with the size of the integers. This is reminiscent of the polynomial time algorithms for linear programming discussed earlier. Also, Collins' algorithm was not shown to parallelize, although enough is now known that a parallel version probably could be developed. Collins' work has been enormously influential in the area.

The next major complexity breakthrough was made by Grigor'ev [13] who developed a decision method for which $E \approx [O(n)]^{4\omega}$. Grigor'ev's bound is doubly exponential only in the number of quantifier alternations. Many interesting problems can be cast as sentences with only a few quantifier alternations. For these, Grigor'ev's result is obviously significant. Like Collins' quantifier elimination method, Girgor'ev's decision method requires integer coefficients and was not proven to completely parallelize.

Slightly incomplete ideas of Ben-Or, Kozen and Reif [1] were completed by Fitchas, Galligo and Morgenstern [9] to construct a quantifier elimination method with arithmetic operation bound

$$(md)^E \text{Cost} \tag{3.5}$$

where $E = 2^{O(n)}$. This provides an arithmetic operation analog of Collins' bit operation bound. When restricted to integer coefficients, the method also yields the Collins' bound if the arithmetic operations are carried out bit by bit. Moreover, the algorithm parallelizes. Assuming each arithmetic operation requires one time unit, the resulting time bound is

$$[E \log(md)]^{O(1)} + \text{Time}(N) \tag{3.6}$$

if $(md)^E N$ parallel processors are used, where Time(N) is the worst-case time required to evaluate the Boolean function holding the atomic predicates using N parallel processors. The analogous time bound for bit operations is also valid, namely, $[E \log(Lmd)]^{O(1)} + \text{Time}(N)$ if $(md)^E[L^{O(1)} + N]$ parallel processors are used.

In [25], the author introduced a new quantifier elimination method for which $E = \prod_{k=0}^{\omega} O(n_k)$. This was established for arithmetic operations and bit operations, i.e., (3.4) and (3.5). Regarding bit operations, the dependence of the bounds on L was shown to be very low, $L^{O(1)}$ in (3.4) being replaced by $L(\log L)(\log\log L)$, i.e., the best bound known for multiplying two L-bit integers. Moreover, the method was shown to parallelize, resulting in the arithmetic operation time bound (3.6) if $(md)^E N$ parallel processors are used, and the bit operation time bound $\log(L)[E \log(md)]^{O(1)} + \text{Time}(N)$ if $(md)^E[L^2 + N]$ parallel processors are used.

Independently and simultaneously, Heintz, Roy and Solerno [15] developed a quantifier elimination method for which $E = O(n)^{O(\omega)}$, both for arithmetic and bit operations. Their method also completely parallelizes.

The various bounds are best understood by realizing that quantifier elimination methods typically work by passing through a formula from back to front. First the vector x_w is focused on, then the vector $x_{\omega-1}$, and so on. Some methods ([5, 25]) make a second pass, from front to back. The work arising from each vector results in a factor for E. For Collins' quantifier elimination method, the factor corresponding to x_k is $2^{O(n_k)}(2^{O(n)} = 2^{O(n_0)} \ldots 2^{O(n_\omega)})$. For the method introduced by the author, the factor is $O(n_k)$. The factor corresponding to Grigor'ev's decision method is $\approx O(n)^4$ independently of the number of variables in x_k. In that method a vector with few variables can potentially create as much work as one with many variables. Similarly, the factor corresponding to the quantifier elimination method of Heintz, Roy and Solerno is $O(n)^{O(1)}$ independently of x_k.

For the record, the quantifier elimination method in [25] produces a quantifier-free formula of the form

$$\bigvee_{i=1}^{I} \bigwedge_{j=1}^{J_i} (h_{ij}(z) \Delta_{ij} 0)$$

where $I \le (md)^E$, $J_i \le (md)^{E/n_0}$, $E = \prod_{k=0}^{\omega} O(n_k)$, the degree of h_{ij} is at most $(md)^{E/n_0}$ and the Δ_{ij} are standard relations $(\ge, >, =, \ne, <, \le)$. If the coefficients of the original formula are integers of bit length at most L, the coefficients of the polynomials h_{ij} will be integers of bit length at most $(L + n_0)(md)^{E/n_0}$.

Results of Weispfenning [39], and Davenport and Heintz [6], show the double exponential dependence on ω of the above bound on the degrees of the polynomials h_{ij} cannot be improved in the worst case.

Fisher and Rabin [8] proved an exponential worst-case lower bound for decision methods. However, the lower bound is exponential only in the number of quantifier alternations, and is only singly exponential in that. A tremendous gap remains between the known upper and lower bounds for decision methods.

In closing this section we mention that work of Canny ([3], [4]) has been especially influential in this area in recent years, both for the techniques he has developed and employed and for the connections he has established between the area and robotics. Work of Vorobjov [14] has also been very influential.

4. Solving Formulae Approximately

In this section we discuss the complexity of approximating solutions for formulae, restricting attention to the field $\mathbb{R}$ of real numbers.

The most basic problem in this vein is that of approximating roots of univariate polynomials. Sequential bounds for this problem are numerous, for various models of computation, and have been proven over many years. Discussions have been provided by Smale ([31, 33]), and Schonhage [29].

Until very recently significant time bounds for the parallel computation of roots of univariate polynomials have been missing. However, Neff [22] has proven, in the parlance of computer science, that the problem is in NC. He has shown that all roots of a univariate polynomial can be approximated to within Euclidean distance $\varepsilon > 0$ in time $O[\log(Ld)+\log\log(4+\frac{1}{\varepsilon})]^3$ using $[Ld\ \log(2+\frac{1}{\varepsilon})]^{O(1)}$ parallel processors, where $d \geq 3$ is the degree of the univariate polynomial and L is the maximal bit length of the coefficients, assumed to be integers. Although Neff does not present an arithmetic operation time-bound for arbitrary real number coefficients, his ideas can be extended to do so. Assuming we desire to approximate all roots lying within distance r of the origin, the resulting time bound is of the form $[\log(d)\log\log(4+\frac{r}{\varepsilon})]^{O(1)}$ if $[d\ \log(2+\frac{r}{\varepsilon})]^{O(1)}$ parallel processors are used, assuming one time unit is required per processor per arithmetic operation.

Neff's result and techniques have implications beyond the univariate setting. For example, the principal bottleneck in parallelizing Collins' quantifier elimination method has been its reliance on univariate polynomial root approximation. Neff's result removes that bottleneck.

In [26], the author reduces the problem of approximating solutions for formulae (3.3) to the problem of approximating roots for univariate polynomials. Both sequential and parallel complexity bounds for the reduction are provided. Using Neff's algorithm and ignoring the cost of evaluating the Boolean function holding the atomic predicates (which generally is a relatively negligible cost), the resulting arithmetic operation time bound is $[E\ \log(md)\log\log(4+\frac{r}{\varepsilon})]^{O(1)}$ if $[(md)^E\ \log(2+\frac{r}{\varepsilon})]^{O(1)}$ parallel processors are used, where as in the last section, $E = \prod_k O(n_k)$. For each connected component of the solution set that intersects $\{z; \|z\| \leq r\}$, a point within Euclidean distance ε of the component is computed within this time, assuming that either (i) both r and ε are input to the algorithm or (ii) only ε is input and r is defined to be the infimum of distances of all solutions from the origin.

The resulting bit operation time bound is $[E\ \log(Lmd)\log\log(4+\frac{1}{\varepsilon})]^{O(1)}$ if $[L\ \log(2+\frac{1}{\varepsilon})]^{O(1)}(md)^E$ parallel processors are used. For each connected component of the solution set, a point within distance ε of the component is computed within this time.

(The solution set consists of at most $(md)^E$ connected components. If the coefficients of the formula are all integers of bit length at most L then each connected component of the solution set has a point within distance $2^{\overline{L}}$ of the origin, where $\overline{L} = L(md)^E$.)

Sequential bounds established in [26], the best presently available, are $(md)^E\log\log(4+\frac{r}{\varepsilon})$ arithmetic operations and $\hat{L}^2\log(\hat{L})\log\log(\hat{L})(md)^E$ bit operations where $\hat{L} = L+\log(2+\frac{1}{\varepsilon})$. The dependence of this arithmetic operation bound on r and ε cannot be improved, as was proven by the author in [24].

The occurrence of the ratio r/ε in both the sequential and parallel arithmetic operation bounds naturally leads one to suspect that analogous bounds hold for relative error, that is, for the problem of computing a point within distance $\varepsilon/\|\bar{x}\|$ of an actual solution $\bar{x}$. However, it is easily proven that even for the problem of computing relative approximations to roots of quadratic polynomials, no real number model algorithm has a uniform arithmetic operation bound independent of the quadratic polynomial, depending only on the degree and relative error desired. Uniform bounds for computing points within specified relative error of a solution require that the basic algorithmic operations include more than just arithmetic operations and comparison operations, e.g., radicals.

5. Ill-Posed Problem Instances

Complexity theory has been developed almost exclusively for problems for which exact data is used in computations. A theory which fully incorporates the use of approximate data has yet to be developed. Central to such a development will be the notion of an ill-posed problem instance. In this section we relate an answer to the question, "Is it possible to know a problem instance is ill-posed?" A complete development can be found in [27]. Here we can only sketch a few of the ideas, rather vaguely.

Define a problem to be a formula $P(x,y)$ with two vectors $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ of free variables, $\mathbb{R}$ denoting the real numbers. A *problem instance* corresponds to specifying values for y. The values are the *data* for the instance. We say "*x is a solution for instance y*" if the pair (x,y) is a solution for the formula.

Many problems can be cast in this format. For example, linear programming with a fixed number of constraints and variables fits this format. The vector y then specifies A, b and c in (2.1).

We assume a formula $P(x,y)$ encoding the problem of interest is known and we assume arbitrarily accurate approximate data for the actual instance is available through an oracle. Input to the oracle is $\delta > 0$ and output is $\bar{y}$ strictly within error δ of the data for the actual instance. (Very general functions are allowed in measuring solution and data errors, but for this brief synopsis assume errors are measured by norms.)

The goals:

1) Determine if the actual instance has a solution.
2) If it has a solution, compute a ε-approximate solution, i.e., $\bar{x} \in \mathbb{R}^n$ guaranteed to be within error ε of a solution for the actual instance.

The goals are to be achieved using approximate data and any other information available about the actual instance, e.g., it might be known the actual instance has infinitely many solutions, even if the exact data for the instance is not known. There are no restrictions on the known information, including its form, except it is required to be consistent, i.e., a contradiction cannot be deduced from it.

Depending on the known information and the exact data for the actual instance, the goals may not be achievable. Roughly, the actual instance is *ill-posed* if the goals cannot be achieved regardless of how accurate the approximate data is.

To be more definite, we introduce three definitions, which are discussed at length in [27].

Definition. A problem instance $\tilde{y}$ is *indistinguishable from the actual instance* if the known information regarding the actual instance does not exclude the possibility that $\tilde{y}$ is the actual instance.

Definition. An *acceptable algorithm* for the problem:

1) Accepts as input any tuple $(\bar{y}, \delta, \varepsilon)$ where $\delta > 0$, $\varepsilon > 0$ and $\bar{y}$ might be provided by the oracle upon input δ.
2) Replies one of the following three statements:
   (a) "All instances which
      (i) are indistinguishable from the actual instance and
      (ii) are strictly within error δ of $\bar{y}$,
      have solutions, and $\bar{x}$ is strictly within error ε of a solution for all such instances," (where $\bar{x}$ is computed by the algorithm).
   (b) "All instances which
      (i) are indistinguishable from the actual instance and
      (ii) are strictly within error δ of $\bar{y}$,
      do not have solutions."
   (c) "Please provide better data accuracy."
3) Can be proven correct, i.e., correct in the sense that whenever it replies with statement (2a) or (2b) and the input tuple satisfies the condition that $\bar{y}$ is strictly within error δ of data for an instance which is indistinguishable from the actual instance then the statement replied is indeed true.

The motivation for requiring that the algorithm can be proven correct in the sense of (3) is that if the algorithm does, say, reply with statement (2a) upon some input for which $\bar{y}$ is strictly within error δ of the actual instance then one can prove, in terms of one's knowledge regarding the actual instance, that the point $\bar{x}$ is indeed within error ε of a solution for the actual instance. In other words, one can be certain in terms of what one knows about the actual instance that the algorithm will not erroneously claim a certain point to be within error ε of a solution for the actual instance when in fact it is not. *Indeed, the reader should regard the definition of an acceptable algorithm to simply formalize the requirement that one be able to trust the algorithm not to reply with an incorrect answer for the actual instance.*

Definition. The actual instance is *definitely well-posed* if some acceptable algorithm replies (2a) or (2b) whenever $\bar{y}$ is strictly within error δ of the actual instance and $\delta > 0$ is sufficiently small, where what constitutes sufficiently small δ may depend on ε.

We do not provide a precise definition of an ill-posed problem instance. We only assume that whatever definition is chosen, it excludes 'definitely well-posed' instances.

In [27], the author argues that if the information known about the actual instance can be used to deduce the actual instance is not 'definitely well-posed' then it can be used to deduce the actual instance is 'definitely well-posed', a contradiction. Consequently, if the known information is consistent, it cannot be known the actual instance is ill-posed. (Somewhat inaccurately, the result amounts to saying that if one knows arbitrarily accurate approximate data is insufficient for the goals, then that knowledge provides sufficient information to design acceptable algorithms for which accurate approximate data is sufficient, a contradiction.) Consequently, for problems corresponding to real formulae it is impossible to know if for the instance of interest it is pointless to collect better approximate data and try new algorithms (although it is certainly possible to sometimes know it is not pointless).

The impossibility of being able to know the actual instance is ill-posed is primarily a consequence of the existence of decision methods for the first order theory of the reals. If instead, for example, solutions are required to be rational vectors, examples can be easily constructed showing it is possible to deduce from the known information that the actual instance is not 'definitely well-posed' without arriving at a contradiction. This is discussed at greater length in [27].

Again, much work needs to be done to develop a complexity theory which incorporates the use of approximate data. Some reflections on this have been provided by Demmel [7], Smale [34] and Shub [30]. In [27], basics for a very general theory of condition numbers are developed.

References

1. Ben-Or, M., Kozen, D., Reif, J.: The complexity of elementary algebra and geometry. J. Comp. System Sci. **32** (1986) 251–264
2. Blum, L., Shub, M., Smale, S.: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. Bull. Amer. Math. Soc. **21** (1989) 1–46
3. Canny, J.: The Complexity of Robot Motion Planning. MIT Press, Cambridge, Mass., 1989
4. Canny, J.: Some algebraic and geometric computations in PSPACE. Proceedings of the 20th Annual ACM Symposium on the Theory of Computing 1988, pp. 460–467
5. Collins, G.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. (Lectures Notes in Computer Science, vol. 33). Springer, Berlin Heidelberg New York 1975, pp. 515–532
6. Davenport, J., Heintz, J.: Real quantifier elimination is doubly exponential. J. Symb. Comp. **5** (1988) 29–35
7. Demmel, J.: On condition numbers and the distance to the nearest ill-posed problem. Num. Math. **51** (1987) 251–289
8. Fischer, M., Rabin, M.: Super-exponential complexity of Presburger arithmetic. In: Complexity of Computations (SIAM-AMS Proc., 7), 1974, pp. 27–41
9. Fitchas, N., Galligo, A., Morgenstern, J.: Algorithmes rapides en sequentiel et en parallel pour l'élimination de quantificateurs en geométrie élémtaire. Séminaire Structures Ordonnées, U.E.R. de Math. Univ. Paris VII, 1987
10. Goldfarb, D., Todd, M.: Linear programming. In: Nemhauser, G., Rinnooy Kan, A., Todd, M. (eds.) Handbooks in Operations Research and Management Science, vol. I: Optimization, Chapter 2. North-Holland, Amsterdam 1989
11. Gonzaga, C.: An algorithm for solving linear programming problems in $O(n^3L)$ operations. In: Megiddo, N. (ed.) Progress in Mathematical Programming – Interior Point and Related Methods, Chapter 1. Springer, Berlin Heidelberg New York 1989
12. Gonzaga, C.: Path following methods for linear programming. To appear in SIAM Review
13. Grigor'ev, D.Yu.: The complexity of deciding Tarski algebra. J. Symb. Comp. **5** (1988) 65–108

14. Grigor'ev, D.Yu., Vorobjov, N.N. Jr.: Solving systems of polynomial inequalities in subexponential time. J. Symb. Comp. **5** (1988) 37–64
15. Heintz, J., Roy, M.-F., Solernó, P.: Sur la complexite du principe de Tarski-Seidenberg. Bull. Soc. Math. France **118** (1990) 101–126
16. Karmarkar, N.: A new polynomial time algorithm for linear programming. Combinatorica **4** (1984) 373–395
17. Khachiyan, L.G.: Polynomial algorithms in linear programming. USSR Computational Mathematics and Mathematical Physics **20** (1980) 53–72
18. Kojima, M., Mizuno, S., Yoshise, A.: A polynomial-time algorithm for a class of linear complementarity problems. Mathematical Programming **44** (1989) 1–26
19. Megiddo, N.: Pathways to the optimal set in linear programming. In: Megiddo, N. (ed.) Progress in Mathematical Programming – Interior Point and Related Methods, Chapter 8. Springer, Berlin Heidelberg New York 1989
20. Megiddo, N.: On the complexity of linear programming. In: Bewley, T. (ed.) Advances in Economic Theory. Cambridge University Press, 1987, pp. 225–268
21. Monteiro, R.C., Adler, I.: Interior path-following primal-dual algorithms. Mathematical Programming **44** (1989) 27–66
22. Neff, A.C.: Specified precision polynomial root isolation is in NC. To appear in J. Comp. System Sci.
23. Renegar, J.: A polynomial-time algorithm, based on Newton's method, for linear programming. Mathematical Programming **40** (1988) 59-94
24. Renegar, J.: On the worst-case arithmetic complexity of approximating zeros of polynomials. J. Comp. **3** (1987) 90–113
25. Renegar, J.: On the computational complexity and geometry of the first-order theory of the reals. To appear in J. Symb. Comp.
26. Renegar, J.: On the computational complexity of approximating solutions for real algebraic formulae. To appear in SIAM J. Computing
27. Renegar, J.: Is it possible to know a problem instance is ill-posed? To appear in: From Topology to Computation, Proceedings of the Smalefest
28. Renegar, J., Shub, M.: Unified complexity analysis for Newton LP methods. To appear in Mathematical Programming
29. Schonhage, A.: Equation solving in terms of computational complexity. Proceedings of the International Congress of Mathematicians, Berkeley, CA, 1986. American Mathematical Society, 1987, pp. 131–153
30. Shub, M.: On the work of Steve Smale on the theory of computation. To appear in: From Topology to Computation, Proceedings of the Smalefest
31. Smale, S.: On the efficiency of algorithms of analysis. Bull. Amer. Math. Soc. **13** (1985) 87–121
32. Smale, S.: Newton's method estimates from data at one point. The Merging of Disciplines in Pure, Applied and Computational Mathematics. Springer, Berlin Heidelberg New York 1986, pp. 185–196
33. Smale, S.: Algorithms for solving equations. Proceedings of the International Congress of Mathematicians, Berkeley, CA, 1986. American Mathematical Society, 1987, pp. 87–121
34. Smale, S.: Some remarks on the foundations of numerical analysis. SIAM Review **32** (1990) 211–220
35. Tardos, E.: A strongly polynomial algorithm for solving combinatorial linear programs. Operations Research **34** (1986) 250-256
36. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. University of California Press, 1951
37. Vaidya, P.: An algorithm for linear programming which requires $O(((m+n)n^2+(m+n)^{1.5}n)L)$ arithmetic operations. Mathematical Programming **47** (1991) 175–202

38. Vaidya, P.: Speeding-up linear programming using fast matrix multiplication. Proceedings of the 30th IEEE Symposium on the Foundations of Computer Science, pp. 332–337
39. Weispfenning, V.: The complexity of linear problems in fields. J. Symb. Comp. **5** (1988) 3–27
40. Ye, Y.: An $O(n^3L)$ potential reduction algorithm for linear programming. To appear in Mathematical Programming