Algorithms, complexity, and the sciences

Christos Papadimitriou[1]

Simons Institute for the Theory of Computing, University of California, Berkeley, CA 94720

This contribution is part of the special series of Inaugural Articles by members of the National Academy of Sciences elected in 2009.

Contributed by Christos Papadimitriou, September 12, 2014 (sent for review July 20, 2014; reviewed by Avi Wigderson and Jon Kleinberg)

Algorithms, perhaps together with Moore's law, compose the engine of the information technology revolution, whereas complexity—the antithesis of algorithms—is one of the deepest realms of mathematical investigation. After introducing the basic concepts of algorithms and complexity, and the fundamental complexity classes P (polynomial time) and NP (nondeterministic polynomial time, or search problems), we discuss briefly the P vs. NP problem. We then focus on certain classes between P and NP which capture important phenomena in the social and life sciences, namely the Nash equlibrium and other equilibria in economics and game theory, and certain processes in population genetics and evolution. Finally, an algorithm known as multiplicative weights update (MWU) provides an algorithmic interpretation of the evolution of allele frequencies in a population under sex and weak selection. All three of these equivalences are rife with domain-specific implications: The concept of Nash equilibrium may be less universal—and therefore less compelling—than has been presumed; selection on gene interactions may entail the maintenance of genetic variation for longer periods than selection on single alleles predicts; whereas MWU can be shown to maximize, for each gene, a convex combination of the gene's cumulative fitness in the population and the entropy of the allele distribution, an insight that may be pertinent to the maintenance of variation in evolution.

lens of computation | complexity of equilibria | multiplicative weights update

Information technology has inundated and changed our world, as it is transforming the ways we live, work, play, learn, interact, and understand science and the world around us. One driving force behind this deluge is quite obvious: Computer hardware has become more cheap, fast, and innovative over the past half century, riding as it does on the exponent of Moore's law (1). Progress in efficient algorithms—methods for solving computational problems in ways that take full advantage of fast hardware—is arguably of even greater importance.

Algorithms have been known since antiquity. In the third century BC Euclid wrote about his algorithm for finding the greatest common divisor of two integers. The French scholar G. Lamé noted in 1845 (2) that Euclid's algorithm is efficient, because it terminates after a number of arithmetic operations that grow proportionately to the length of the input—what we call today the number of bits of the two numbers. [In fact, one of the very few works on the subject of algorithms that have been published in PNAS is a 1976 article by Andrew Yao and Donald Knuth, revisiting and refining that analysis (3).] In the ninth century CE, the Arab mathematician Al Khwarizmi codified certain elementary algorithms for adding, dividing, etc., decimal numbers—the precise algorithms we learn today at elementary school. In fact, these simple and powerful algorithms were a major incentive for the eventual adoption of the decimal number system in Europe (*ca*. 1500 CE), an innovation that helped precipitate a social and scientific revolution comparable in impact to the one we are living in now.

The study of efficient algorithms—algorithms that perform the required tasks within favorable time limits—started in the 1950s, soon after the first computer, and is now a very well-developed mathematical field within computer science. By the 1960s, researchers had begun to measure algorithms by the criterion of polynomial time, that is, to consider an algorithm efficient, or satisfactory, if the total number of operations it performs is always bounded from above by a polynomial function (as opposed to an exponential function) of the size of the input. For example, sorting n numbers can be done with about $n \log n$ comparisons, whereas discovering the best alignment of two DNA sequences with n nucleotides can take in the worst case time proportional to n^2 (but can be performed in linear time for sequences that do align well); these are both considered "satisfactory" according to this criterion.

Search Problems

What are the limits of computation? In his 1936 paper that is considered the founding act of computer science (4), Alan M. Turing showed that there are perfectly well-defined computational tasks for which no algorithms are possible (example: Will a given program loop forever?). Computer scientists have been aware of this fundamental limitation of computation from the very start and became used to steering clear of it. However, during the 1960s, researchers in algorithms started encountering "impossible" problems of a different nature. These problems were search problems, in that they shared the following form: Given an input, call it x, find a solution y such that x and y stand in a particular relation to each other that is easy to check. Some of these problems can be solved satisfactorily by algorithms, whereas others seem to require exponential time for their solution. Here are a few illustrious examples of both kinds:

i) Shortest path: Given the pairwise distances in miles between n cities, find a path from the first city to the last city that is shorter than M miles (a given integer), or report that none exists. (An optimization problem, such as this and several other problems in this list, can be transformed into an equivalent search problem, by supplying a limit; thus, the study of search problems encompasses optimization.)

ii) Traveling salesman problem: Given the pairwise distances in miles between n cities, find a path from the first city to

Significance

The theory of algorithms and complexity, of which the basic concepts and results are being reviewed here, is an important and consequential domain of mathematical investigation that is unfamiliar to most PNAS readers. Furthermore, in the second half of the article we focus on recent and current research, applying these concepts and results to help elucidate certain central theoretical problems in the social and life sciences, including market equilibria and the theory of evolution.

Author contributions: C.P. designed research, performed research, contributed new reagents/analytic tools, analyzed data, and wrote the paper.

Reviewers: A.W., Institute for Advanced Study; and J.K., Cornell University.

The authors declare no conflict of interest.

See Profile on page 15858.

[1]Email: christos@berkeley.edu.

This article contains supporting information online at www.pnas.org/lookup/suppl/doi:10.1073/pnas.1416954111/-/DCSupplemental.

the last city, visiting all other $n-2$ cities on the way, that is shorter than M miles, or report that none exists.

iii) Partition: Given a list of whole numbers, for example 112, 54, 224, 156, 87, 34, 171, 114, 56, and 162, find a way to split them into two parts that have the same sum, or report that no such splitting exists.

iv) Partition into pairs: Given a list of n whole numbers, say the 10 numbers 112, 54, 224, 156, 87, 34, 171, 114, 56, and 162, find a way to split them into $n/2$ pairs so that all pairs have the same sum, or report that none exists.

v) Clique: Given a large network—perhaps the friendship graph of a social network connecting 1,000,000 people—find a clique of m points (that is, m people, any two of whom are friends), or report that none exists.

vi) Min-cut: Given the same large network, find a *cut* of m points, that is, m points whose removal would disconnect the network, or report that none exists.

vii) Hamilton path: Given a large network, find a path through the network that visits all points without ever repeating a point, or report that none exists.

viii) Euler path: Given a large network, find a path that visits all lines without ever repeating a line, or report that none exists.

ix) Linear programming: Given a set of m linear inequalities in n variables, find a vector of n reals satisfying them all, or report that none exists.

x) Integer linear programming: Given a set of m linear inequalities in n variables, find a vector of n integers satisfying them all, or report that none exists.

Nondeterministic Polynomial Time, Polynomial Time, and Nondeterministic Polynomial Time Completeness

There is an important mystery clouding search problems: Of the 10 problems presented above, 5 (problems *i*, *iv*, *vi*, *viii*, and *ix*) can be solved by efficient—that is to say, polynomial time—algorithms, and so can a plethora of other search problems. In stark contrast, the remaining 5 problems (problems *ii*, *iii*, *v*, *vii*, and *x*, but many others as well) seem to be "immune" to efficient algorithms. Some of these problems have been attacked for decades by algorithms researchers, albeit without success. Naturally, all of these 10 problems, as well as all search problems, can be solved in exponential time by exhaustively enumerating all possible solutions for the given input.

A momentous question arises: Are there search problems that cannot be solved in polynomial time? Or can all search problems, however complex, be somehow solved efficiently? Can exhaustive search always be telescoped? Most computer scientists strongly suspect that the answer is "no" (see ref. 5 for a recent expository book on this subject). That is, we believe that the dividing line illustrated in the case of the 10 search problems above is real, and problems *ii*, *iii*, *v*, *vii*, and *x* cannot be solved in polynomial time.

In computer science one argues about the capabilities and limitations of algorithms by considering classes of problems and contemplating the relationships between such classes. For example, **NP** denotes the class of all search problems. (The historical term **NP** stands for "nondeterministic polynomial time." Nondeterminism is a fictitious ability of a computing machine to guess correctly, a formalism that had origins in automata theory and was quite mainstream in computer science during the 1960s and 1970s. Nondeterministic polynomial time delimits precisely the search problems, because a machine with this ability, given an input, can guess the correct solution, if it exists, and verify it, all in polynomial time.) In contrast, **P** (polynomial time) denotes the class of all search problems that can be solved efficiently. Looking back at the 10 search problems, they all lie in **NP**, but only problems *i*, *iv*, *vi*, *viii*, and *ix* are known to be in **P**.

One can now state the widespread suspicion that not all search problems can be solved in polynomial time as a mathematical conjecture:

Conjecture 1. $\mathbf{NP} \neq \mathbf{P}$.

Proving this important conjecture has eluded computer scientists and mathematicians for almost half a century and remains one of the deepest and most fundamental open problems in mathematics, and science more broadly, today.

During the long wait for the proof of this grand conjecture, almost all computer scientists have been proceeding with the working hypothesis that $\mathbf{NP} \neq \mathbf{P}$, and therefore certain search problems must require exponential time. However, exactly which problems require exponential time? And how can one tell?

A major advance in this front has been a strikingly comprehensive answer to these questions:

Theorem 1. *Problems ii, iii, v, vii, and x above are all equivalent, in the following sense: Either they can all be solved in polynomial time or none of them can. Furthermore, they can be solved in polynomial time if and only if* $\mathbf{NP} = \mathbf{P}$.

This result was proved in 1972 by Richard Karp (6), building on work by Stephen Cook (7); Leonid Levin (8) had independently arrived at similar results. Search problems that have this property (that is, can be solved in polynomial time only if $\mathbf{NP} = \mathbf{P}$) are called **NP**-complete.

To establish that a particular problem is **NP**-complete entails the use of a specialized kind of algorithm called a reduction: A reduction R from search problem A to search problem B is a polynomial algorithm R, which, given an input x to problem A, transforms it into an equivalent input $R(x)$ to problem B. "Equivalent" here means that any solution of $R(x)$ can be easily transformed to a solution of the original input x to problem A.

Total Search Problems

It has been known almost since the inception of **NP**-completeness that, in principle, there are many search problems that lie between **P** and the **NP**-complete problems (9)—unless, of course, $\mathbf{NP} = \mathbf{P}$, in which case there is nothing between **P** and the **NP**-complete problems. And yet the vast majority of search problems one encounters in practice can be classified as either belonging to **P** or being **NP**-complete. During the past four decades, literally thousands of search problems originating from diverse areas of computer science; mathematics; engineering; and the natural, social, and life sciences have been shown through appropriate reductions to be **NP**-complete, and therefore, presumably, intractable, whereas many others have been successfully solved by polynomial time algorithms. This methodology for classifying search problems has been a key ingredient of our computational way of understanding the world.

Only a few search problems have resisted such classification; most of them are of a variety known as "total." A total search problem is a search problem for which it is known a priori that at least one solution of the kind sought always exists, for any possible input—and hence the only challenge is to find it. One important example of a such a problem is Factoring: Given an integer, find its factorization into prime factors; once such a list of prime factors is provided, it can be readily checked in polynomial time whether they are indeed prime (10) and whether their product is the input integer. The search problem Factoring is total because a solution (prime factorization) exists for any input (integer). It is a well-known difficult problem—in fact, widely used cryptographic systems are based on the presumption that it is intractable (11). And yet, it is not known to be **NP**-complete, and in fact researchers do not expect it to be. The reason is that, informally, **NP**-complete problems such as Clique draw much of their difficulty from the fact that a solution may fail to exist (a network may or may not possess a clique of the

desired size). As a result, there seems to be no way to reduce an **NP**-complete problem such as Clique to a total problem such as Factoring. A completely different way is needed to establish the difficulty of hard total search problems.

During the 1980s and 1990s, a principled way was developed for classifying total search problems in terms of their proofs of totality. Every total search problem is in essence an existence theorem stating that, for every input, there exists a solution that stands in a special relationship to that input, and this existence theorem must possess a proof. Unless the total search problem is easily solvable in polynomial time, this existence proof must contain at least one particular step that is "not polynomially constructive," typically a simple combinatorial argument on an exponentially large object such as a graph. Thus, one can group together search problems according to the nature of this nonconstructive step in their proofs of totality. As it turns out, there are a fairly small number of nonconstructive steps one encounters in the known total search problems, giving rise to a few complexity classes lying somewhere between **P** and **NP**: These classes are known as polynomial parity argument (**PPA**), the polynomial pigeonhole principle (PPP), polynomial parity argument for directed graphs (**PPAD**), and polynomial local search (**PLS**); see, for example, ref. 12.

One of these nonconstructive arguments is the **PPAD**. Recall that a directed graph is a finite set of points connected by arrows. A point in this graph is balanced if the number of outgoing arrows equals the number of incoming arrows; otherwise it is unbalanced. Suppose now that one is presented with a directed graph, and one notices that one of its points is unbalanced. It is clear that there must be another unbalanced point somewhere in the graph—simply because the sum of incoming arrows over all points must equal the sum of outgoing arrows. The question is, How does one find this other unbalanced point? Naturally, one can inspect every other point until one discovers a point that is unbalanced. However, what if this graph is huge? What if it is an exponentially large graph, and the arrows are provided by an algorithm?

This is the essence of the class **PPAD**: search problems whose existence of solution is guaranteed by virtue of an exponentially large directed graph, implicitly represented through an algorithm, plus one given unbalanced point of the graph (formal definition in *SI Text*). Many problems are known to be **PPAD**-complete; perhaps the most fundamental such problem is related to Brouwer's theorem: Every continuous function from a convex, compact set to itself has a fixed point. The question is, If one is given such a function, how does one find this fixed point, even approximately? This problem was shown in ref. 12 to be **PPAD**-complete. As we shall see in the next section, fixed-point arguments lie in the basis of fundamental results in economics.

The class **PLS**, on the other hand, is a genre of total search problems whose proof of totality is based on another elementary property of directed graphs: Either a finite directed graph has a cycle or else it must have a sink, a point with no outgoing arrows. The reason is again elementary: If there is always an arrow out of every point we encounter, then by following arrows we must sooner or later close a cycle (because the graph is finite). The absence of cycles can be guaranteed by a potential function p mapping points to the real numbers. If every arrow from point a to point b satisfies $p(a) > p(b)$, then clearly a cycle cannot exist. A problem in **PLS** is presented by such a directed graph, again implicitly represented by an algorithm that encodes the arrows and the function p (precise definitions in *SI Text*). Any sink is sought. This class captures local energy minima in physics or local maxima of fitness landscapes in evolution. Whereas physicists and evolutionary biologists often speak about a process "getting stuck" at a local optimum, from the algorithmic point of view it turns out that, by all available evidence, getting stuck at a local optimum may be an exponentially difficult task.

We are now ready to articulate two hypotheses that are our point of departure:

Conjecture 2. PPAD ≠ P.

Conjecture 3. PLS ≠ P.

These two conjectures are stronger than *Conjecture 1*, in that they may fail to hold even if **NP ≠ P**. The reasons why we believe them true are similar to the grand conjecture: Many researchers have tried hard for a long time to develop polynomial time algorithms for many problems in these classes, albeit without success. Besides, exponential lower bounds for very natural kinds of algorithms for these problems are actually known (13, 14). And it seems counterintuitive that there is always a way to telescope the search through all points of an exponentially large directed graph, to zero in the other unbalanced node, or the sink, that is sought.

Complexity in Economics

In the balance of this paper we focus on one particular mode of application of the theory of algorithms and complexity reviewed in the previous sections, namely on its role in enhancing our understanding of phenomena and problems in the natural, social, and life sciences; this role has often been termed "the lens of computation." In this section we consider game theory and economics.

Games. Games are mathematical thought experiments used in the study of rational strategic behavior in the social sciences; here we briefly recall the basic concepts and notation of the theory of games, which are useful in this and the next section. A game consists of a finite number of players; for each player, a finite set of actions or strategies; and a utility function assigning to each choice of actions, one by each player, a real number corresponding to the gain obtained (or loss suffered, if negative) by player i if the players choose these actions. For example, the game shown below involves two players, the row player R and the column player C, each with two strategies. This game is of a simple kind known as two-player zero-sum games; the utilities shown are those of R, whereas the utility of C at each action combination (entry of the table) is the negative of that shown. In other words, once the two players choose an action, the column player pays the row player the amount shown:

3	−1
−2	1

What would two rational players do in this situation? Game theory has a rather sharp prediction for two-player zero-sum games, first stated by John von Neumann in 1928 (15): In this particular example, it turns out that R will play row 1 with probability $x_1 = 3/7$ and row 2 with probability $x_2 = 4/7$. This is her maxmin mixed strategy, a randomized action that "exposes her" as little as possible to C's minimizing choice. Player C will play with probabilities $y_1 = 2/7$, $y_2 = 5/7$ (his minmax strategy). The value of this game—that is to say, the expected amount of money that will change hands if both players play as predicted—can now be easily calculated to 1/7.

How does one compute the maxmin/minmax pair of a large zero-sum game? It has been known for a long time that this can be done through linear programming (16); see also ref. 17 for the latest on this equivalence. Here we explain a very simple and intuitive alternative method, known as the multiplicative weight update (MWU) algorithm: Imagine that the two players play the game shown above repeatedly, say starting with the uniform distribution where all strategies are played with probability 1/2. Suppose that, at each repetition, R computes the expected

payoff of each of her two strategies and then boosts the probabilities of strategies that do better than average, while diminishing the probabilities of the others, whereas C plays his best response to that. Concretely, assume that, if player R plays her ith strategy at repetition t with probability x_i^t and gets from playing it expected payoff U_i^t (a positive or negative number, a linear function of the opponent's probabilities), then R will play next time the ith strategy with probability x_i^{t+1} given by the formula

$$x_i^{t+1} = \frac{1}{Z^t} x_i^t \cdot (1 + \epsilon \cdot U_i^t), \qquad [1]$$

where $\epsilon > 0$ is a small parameter and $Z^t = \sum_i x_i^t \cdot (1 + \epsilon \cdot U_i^t)$ is a normalizing term selected to keep the probabilities adding to one. Player C chooses the strategy with smallest expectation:

t	1	2	3	10	20	29	30
x_1^t	0.4950	0.4900	0.4850	0.4502	0.4352	0.4253	0.4375

We show in the above table a few repetitions of MWU with $\epsilon = .01$. Note that the process seems to converge to R's maxmin strategy $x_1 = 3/7 \approx 0.4285$. Indeed, MWU, a seemingly very primitive and common-sense method, can be proved to solve this problem (along with many other more sophisticated problems in computer science; for example, ref. 18). (However, MWU is not, strictly speaking, a polynomial time algorithm for the two-player zero-sum game problem according to our criteria, but a compromise termed the polynomial time approximation scheme.)

We have so far discussed two-player zero-sum games, one of the simplest kinds of games. Coordination games are even simpler: In a coordination game the utilities of the players are identical (instead of opposite). That is, whereas zero-sum games model situations of total conflict, in a coordination game there is no conflict whatsoever, and the players need only to agree on an advantageous action combination. Such games are interesting when the players cannot communicate or are cognitively weak; they turn out to be important in the next section, when games are played by genes.

Nash Equilibrium. In games with more than two players, or games with two players whose utilities are not exactly opposite, von Neumann's theorem no longer applies and a minmax/maxmin mixed strategy pair is not guaranteed to exist. In 1950, John F. Nash introduced a somewhat weaker prediction, which, however, is also very stable and compelling and turned out to be extremely influential: A Nash equilibrium in a game is a mixed strategy by each player, such that none of the players can attain a better expected payoff by changing the corresponding mixed strategy. Note that the minmax/maxmin pair of a zero-sum game is a Nash equilibrium, and so Nash equilibria generalize maxmin/minmax pairs. However, and in contrast to the situation in zero-sum games, a game may have several distinct Nash equilibria resulting in different payoffs for the players.

The Nash equilibrium is the central solution concept (that is to say, prediction of behavior) in game theory, the golden standard against which all others are judged (for example, the discussion in ref. 19, chap. 12). Furthermore, as Roger Myerson argued (20), the Nash equilibrium and the conception of rationality that it entails lie at the foundations of all of modern economic thought. Remarkably, John F. Nash showed in 1951 (21) the following important result, considered by many the beginning of modern game theory:

Theorem 2. *All games have a Nash equilibrium.*

Before Nash's result, only two-person zero-sum games were known to be so endowed, by virtue of von Neumann's minmax theorem, as we have seen. The universality imputed by *Theorem 2* is a crucial part of the importance and impact of the Nash equilibrium: Again according to Myerson (22), no solution concept can be taken seriously if it is vacuous for some games.

It is natural then to ask whether one can compute a Nash equilibrium for a given game and do so efficiently, that is, in time polynomial in the game's description. Because groups of players are supposed to attain this pattern of play in real life, there should be a way to simulate the process by a computer in less than astronomical time. Unfortunately, unlike in two-player zero-sum games, there is no known polynomial time algorithm for finding a Nash equilibrium in a general game; for example, the MWU does not converge to a Nash equilibrium in games that are not zero sum (23). Researchers have been trying for decades to develop algorithms for this problem [in fact, the first paper published in PNAS bearing the term "algorithm" in its title is an article by the mathematician H. W. Kuhn in 1954—not too long after Nash's theorem—proposing an algorithm for this problem, which, however, turned out to be exponential (24)], and yet the question has been open: Can this important object be computed efficiently?

The following relatively recent result (25) has devastating implications in this regard:

Theorem 3. *Finding a Nash equilibrium for a given game is* ***PPAD****-complete.*

Thus, if one assumes *Conjecture 2*, there are families of games for which finding a Nash equilibrium requires time growing faster than any polynomial—that is to say, games in which the equilibrium promised by Nash's theorem is not realistically accessible. The important universality property of the Nash equilibrium becomes suspect on computational grounds. In fact, *Theorem 3* holds even if there are only two players (26) or if one wants to find only an approximate Nash equilibrium; finding an exact equilibrium is even harder (27). Nash proved his theorem by invoking Brouwer's fixed-point theorem; the **PPAD**-completeness proof in ref. 25 goes in the opposite direction, in that it exhibits a reduction from the problem of finding a Brouwer fixed point to that of finding a Nash equilibrium; see ref. 28 for an accessible exposition of the result and the proof.

Market Equilibria and Complexity. Nash's 1951 proof provided inspiration for a celebrated result in economics, the Arrow–Debreu theorem (29), establishing the existence of price equilibria in economies with perfect competition, thus resolving a fundamental problem first articulated almost a century before by Walras (30). A price equilibrium is a set of prices, one for each good, which, intuitively, motivates all producers and all consumers to behave in such a way that the market clears (there is no excess supply or excess demand). It implies that in such economies there are allocations that are Pareto efficient, a coveted concept of optimality in economics, meaning that there is no other allocation in which all agents fare better. Like Nash's, the proof of the Arrow–Debreu theorem also relies on Brouwer's theorem [more precisely, on a variant known as Kakutani's theorem (31)].

Again, no efficient algorithms are known for actually finding these prices, despite important algorithmic work by Herbert Scarf (32) and many others. Recently the following result was shown (33):

Theorem 4. *Finding equilibrium prices in an economy is* ***PPAD****-complete even for an exchange economy (no production) with concave constant elasticity of substitution consumer utilities (a class of simple and well-behaved utility functions).*

The Arrow–Debreu theorem assumes an economy of perfect competition between producers as well as consumers of goods. It also assumes that both consumer preferences and production constraints have a crucial convexity property. Convexity of consumer preferences is a reasonable assumption, as it predicts that

a consumer's appetite is saturated with increased consumption. In contrast, the convexity assumption in production is unrealistic, because it rules out a feature of most industries and production technologies, namely economies of scale. In markets exhibiting economies of scale in production—that is to say, in any realistic market—the Arrow–Debreu theorem does not hold: A price equilibrium, or a Pareto-efficient allocation, may fail to exist.

By using the theory of **NP**-completeness, it was shown in ref. 34 that such markets may harbor a different—sinister—kind of equilibrium: A complexity equilibrium is an allocation in the economy (production levels for the industries, consumption levels for the consumers) that clears the market, but has the following inauspicious properties: (*i*) the allocation is not Pareto optimum, in that there is a better allocation in which all agents have much better utility; (*ii*) however, finding this or any other better allocation is an **NP**-complete problem—and therefore, unless $\mathbf{P}=\mathbf{NP}$, no market mechanism, price or otherwise, can discover it.

Theorem 5. *There are infinite families of markets with convex consumer utilities, but with economies of scale in production, which have complexity equilibria.*

Finally, in ref. 35 a complexity result is shown concerning price adjustment mechanisms in an exchange economy (the special case of a marketplace without production). A price adjustment mechanism is any function that sets the prices of goods by taking into account the observed excess demand, but also the past history of prices and excess demands. There is an extensive literature proposing price mechanisms that eventually attain a price equilibrium (36–40). The results in ref. 35 establish that such mechanisms cannot converge in any feasible number of iterations. In particular, the following is shown, among other results in the same vein:

Theorem 6. *For any price adjustment mechanism $\mathcal{A}$ and any function f from the reals to the integers, there is an exchange economy with three goods and a unique price equilibrium on which $\mathcal{A}$ takes more than $f(\epsilon)$ iterations to find prices that are within ϵ of the price equilibrium.*

Note that, unlike all other complexity results presented here, *Theorem 6* is an *unconditional* complexity result, in that it does not rely on complexity assumptions such as *Conjecture 1*.

Algorithms, Complexity, and the Theory of Evolution

Evolution is today a powerful, mature, and comprehensive theory, articulating rigorously many of Darwin's brilliant ideas (41) and informed not only by a deluge of molecular data, but also by modern population genetics, mathematics, and statistics; see, for example, ref. 42. Still, there are several important facts about evolution that lack a thorough explanation within the theory. For example, (*i*) sex and recombination are nearly ubiquitous in life, even though they bring concrete disadvantages to individuals and populations (such as the dilution of genetic inheritance from each parent and the breakup of successful combinations), and there is no agreement on how these disadvantages are counterbalanced (43, 44); and (*ii*) current theory predicts that heterozygosity (the probability that two genomes will differ at a particular genetic locus) should be small and roughly proportional to the effective population size, a prediction not confirmed by genetic evidence (45). We next review recent and current research pointing out some interesting connections between the theory of algorithms and complexity and the theory of evolution, which may have some relevance to these questions.

Evolution as an Algorithm. One of the cornerstones of the modern theory of evolution is the mathematical model proposed by Ronald Fisher and Sewall Wright almost a century ago and now broadly established as the standard mathematical formalism of evolution (46). It assumes a population of genotypes, in which the genotype frequencies evolve under selection. The latter is modeled by a fitness function that assigns to each genotype g a nonnegative real number w_g, its fitness, standing for the expected number of offspring by individuals of this genotype. We make here some common assumptions that simplify the mathematics: infinite population, discrete generations, random mating, and free recombination. Finally, we assume a haploid population; we briefly discuss diploidy later. It is then straightforward to write equations that calculate the genotype frequencies in the next generation from the genotype frequencies in the current generation.

Let us further assume that the fitness coefficients w_g for genotype g are of the form $w_g=1+s\delta_g$ for some very small positive number $s \ll 1$ called the selection strength and δ_g s satisfying $|\delta_g| \le 1$; we call δ_g the differential fitness of genotype g. This weak selection assumption is the mathematical articulation of the broadly held belief by evolutionary biologists that selection forces are typically small, and thus evolution is nearly neutral (47). In weak selection, a theorem due to Nagylaki (48) establishes that evolution proceeds within $O(s)$ of linkage equilibrium, that is, the regime in which the genotype frequencies are a product distribution: The frequency f_g of genotype $g=(j_1,\ldots,j_m)$, where m is the number of genes, is equal to $\prod_{i=1}^{m} x_i(j_i)$, where $x_i(j)=\sum_{j_k,k\neq i} f_{(j_1,\ldots j_m)}$ is the frequency of allele i of the first gene in the population. It is shown in ref. 49 that, suppressing $O(s)$ additive terms (recall our assumption that $s \ll 1$), the equation governing the change in the frequency $x_i(j)$ of the allele j of gene i in the population from generation t to generation $t+1$ becomes

$$x_i^{t+1}(j)=\frac{1}{X^t}x_i^t(j)\left(1+\epsilon\Delta_i^t(j)\right), \qquad [2]$$

where $\Delta_i^t(j)$ is the expected differential fitness among genotypes that contain allele j at locus i.

Note now the remarkable similarity between Eq. **2** and the multiplicative updates algorithm in Eq. **1**:

Theorem 7. *Under weak selection and as $s \to 0$, the Fisher–Wright equations become identical to the equations of a repeated coordination game, in which the players are the loci, rounds of play correspond to generations, the actions available to each player/locus are its alleles, the current probability of play of an allele by its locus is the frequency of the allele in the population at the current generation, the utility of each locus is the organism's fitness, and the probabilities of play are updated by each locus at each generation by the MWU algorithm, where the parameter ϵ is equal to the selection strength s.*

This equivalence, pointed out recently in ref. 49, has some intriguing implications. First, MWU is well known in computer science to be a simple, common-sense, and yet astonishingly powerful algorithm, able to solve many sophisticated problems (18). Therefore, it is quite telling that it appears to capture the evolution of the genetic statistics of populations under sex. Second, as pointed out in ref. 49 there is a dual view of MWU as an algorithm through which, at each generation, each locus acts as if it were seeking to maximize the sum of two objectives: One is the cumulative expected fitness over all past generations, and the other is the entropy of the current allele distribution of the locus. In other words, maintenance of genetic diversity is a large part of what this process seems to be about. (Of course, at equilibrium all diversity will be lost; however, it is quite intriguing that its maintenance is center stage in this interpretation.) Finally, because it is recombination that brings MWU into play, this point of view may be of some relevance to the role of sex in evolution.

What if the organism in question is diploid (like all of us)? The MWU point of view is still valid, but the dual interpretation changes in an interesting direction: Each gene seeks to optimize the

sum of three terms. The first two are, as before, the population's cumulative expected fitness and the entropy of the gene's allele distribution. The third term is new: the population's expected cumulative fitness conditioned on this gene being heterozygous, that is, having two different alleles.

Selection on Allele Combinations Across Loci. We conclude with a brief account of current work with Costis Daskalakis and Adi Livnat relating the speed of allele fixation with the complexity class **PLS** defined in *Total Search Problems* (more detailed exposition in *SI Text*). Continuing in the model of the previous section, in the literature it is often assumed further that the loci contribute additively to fitness. This means, if for simplicity we assume that there are two loci, that there are numbers g_i and h_j associated with the alleles of the two loci in our model (standing for the contribution to fitness of allele i of the first locus and of allele j of the second, respectively) so that $f_{ij} = g_i + h_j$. Similarly for more loci, this assumption of additivity implies that selection acts on individual alleles and simplifies the analysis of evolution considerably. For example, it follows easily from additivity that, if the allele i of a locus has the highest fitness contribution h_i, with an advantage of Δ over the second-highest allele fitness, then allele i will be fixed after $O(1/\Delta)$ generations. This calculation is used often in the study of evolution.

Sewall Wright believed that combinations of alleles are key to evolution, whereas recent consideration of the nonaccidental nature of mutations suggests that selection on allele combinations may be a more realistic model of evolution (50). To go one small step beyond the additivity assumption, let us assume that the fitness of a genotype is the sum of fitness contributions associated not with single loci, but with pairs of loci (precise formulation of the model is in *SI Text*). The following can be shown [a similar result was shown independently by Artem Kaznatcheev (51)]:

Theorem 8. *Under Conjecture 3, there are fitness functions that are additive on pairs of loci such that alleles are fixed only after a number of generations that grow faster than any polynomial in the number of loci.*

Thus, variation may persist in selection exponentially longer than predicted by arguments based on individual gene contributions to fitness. In view of this result, it is of interest to determine how "common"—likely to arise in nature—are fitness functions such as these, and this is at present an interesting open problem.

Discussion

Computation is barely seven decades old, and yet its spirit is much older. Latent computational processes underlie much of the world—and certainly the models we have built over the centuries for understanding it. The methodology, techniques, and mindset developed by computer scientists during the past century for the express purpose of grasping the capabilities and limitations of computers have yielded unexpectedly apt insights in a broad spectrum of scientific endeavors. Phase transitions in statistical physics have been usefully linked to the speed of convergence of certain randomized algorithms (52), and even quantum mechanics are presently revisited and revised with computation in mind (53). Here we focused on how the theory of computational complexity developed over the past four decades to help us comprehend why so many practical problems seem to be resistant to efficient solution by computer can be relevant to important questions in the social and life sciences, such as the meaning of equilibria in economics and game theory,and the paradox of heterozygozity in evolution. Further, the evolution of the allele frequencies in a population can be reinterpreted as a game played by genes through a powerful and well-known learning algorithm. I expect that computational insights will continue to prove relevant and useful to more areas of science, even to the scientific field that motivated the beginning of computation (4) and whose object of study is undoubtedly the most explicitly computational of all: understanding the brain.

ACKNOWLEDGMENTS. The author's research has been supported by the National Science Foundation under Grant CCF-0964033 and by the Templeton Foundation under Grant 39966.

1. Mack CA (2011) Fifty years of Moore's Law. *IEEE Trans Semicond Manuf* 24(2): 202–207.
2. Lamé G (1844) Note sur la limite du nombre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers. [Note on the limit of the number of divisions in the search for the greatest common divisor of two integers.] *C R Acad Sci Paris* 19:867–870.
3. Yao AC-C, Knuth DE (1975) Analysis of the subtractive algorithm for greatest common divisors. *Proc Natl Acad Sci USA* 72(12):4720–4722.
4. Turing AM (1936) On computable numbers with an application to the Entscheidungsproblem. *Proc Lond Math Soc* 42(2):230–265, and correction (1937) 43:544–546.
5. Fortnow L (2013) *The Golden Ticket: P, NP, and the Search for the Impossible* (Princeton Univ Press, Princeton).
6. Karp RM (1972) Reducibility among combinatorial problems. *Complexity of Computer Computations*, eds Miller RE, Thatcher JW (Plenum, New York), pp 85–103.
7. Cook SA (1971) The complexity of theorem proving procedures. *Proceedings of the 3rd ACM Symposium on Theory of Computing* (ACM, New York), pp 151–158.
8. Levin L (1973) Universal search problems. *Problemy Peredachi Informatsii* 9(3): 265–266.
9. LadnerRE (1975) On the structure of polynomial time reducibility. *J ACM* 22(1):155–171.
10. Agrawal M, Kayal N, Saxena N (2004) Primes is in P. *Ann Math* 160:781–793.
11. Rivest RL, Shamir A, Adleman LA (1978) A method for obtaining digital signatures and public key cryptosystems. *Commun ACM* 21:120–126.
12. Papadimitriou CH (1994) On the complexity of the parity argument and other inefficient proofs of existence. *J Comput Syst Sci* 48(3):498–532.
13. Papadimitriou CH, Schäffer AA, Yannakakis M (1990) On the complexity of local search. *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing* (ACM, New York), pp 438–445.
14. Hirsch MD, Papadimitriou CH, Vavasis SA (1989) Exponential lower bounds for finding Brouwer fix points. *J Complexity* 5(4):379–416.
15. von Neumann J (1928) Zur Theorie der Gesellschaftsspiele. [Problems of Information Transmission.] *Math Annalen* 100:295–320.
16. Dantzig GB (1963) *Linear Programming and Extensions* (Princeton Univ Press and RAND Corp, Princeton).
17. Adler I (2013) The equivalence of linear programs and zero-sum games. *Int J Game Theory* 42(1):165–177.
18. Arora S, Hazan E, Kale S (2012) The multiplicative weights update method: A meta-algorithm and applications. *Theory Comput* 8(1):121–164.
19. Kreps DM (1990) *A Course in Microeconomic Theory* (Princeton Univ Press, Princeton).
20. Myerson RB (1999) Nash equilibrium and the history of economic theory. *J Econ Theory* 37:1067–1082.
21. Nash JF (1951) Non-cooperative games. *Ann Math* 54:286–295.
22. Myerson RB (1997) *Game Theory: Analysis of Conflict* (Harvard Univ Press, Cambridge, MA).
23. Daskalakis C, Frongillo R, Papadimitriou C, Pierrakos G, Valiant G (2010) On learning algorithms for Nash equilibria. *International Symposium on Algorithmic Game Theory* (Springer, Berlin), pp 114–125.
24. Kuhn HW (1961) An algorithm for equilibrium points in bimatrix games. *Proc Natl Acad Sci USA* 47(10):1657–1662.
25. Daskalakis C, Goldberg PW, Papadimitriou CH (2009) The complexity of computing a Nash equilibrium. *SIAM J Comput* 39(1):195–259.
26. Chen X, Deng X (2006) Settling the complexity of two-player Nash equilibrium. *Proceedings of the Symposium on the Foundations of Computer Science* (IEEE, New York), pp 261–272.
27. Etessami K, Yannakakis M (2007) On the complexity of Nash equilibria and other fixed points (extended Abstract). *Proceedings of the Symposium on the Foundations of Computer Science* (IEEE, New York), pp 113–123.
28. Daskalakis C, Goldberg P, Papadimitriou C (2009) The complexity of computing a Nash equilibrium. *Commun ACM* 52(2):89–97.
29. Arrow K, Debreu G (1954) Existence of an equilibrium for a competitive economy. *Econometrica* 22(3):265–290.
30. Walras L (1874) *Eléments d'Economie Politique*. [Elements of Political Economy] (L Corbaz, Lausanne, Switzerland).
31. Kakutani S (1941) A generalization of Brouwer's fixed point theorem. *Duke Math J* 7: 457–459.
32. Scarf HE (1967) The approximation of fixed points of a continuous mapping. *SIAM J Appl Math* 15(5):1328–1343.
33. Chen X, Paparas D, Yannakakis M (2013) The complexity of non-monotone markets. *Proceedings of the 45th annual ACM Symposium on Theory of Computing* (*STOC*) (ACM, New York), pp 181–190.

34. Papadimitriou CH, Wilkens CA (2011) Economies with non-convex production and complexity equilibria. *Proceedings of the 12th ACM Conference on Electronic Commerce (EC11)* (ACM, New York), pp. 137–146.
35. Papadimitriou CH, Yannakakis M (2010) An impossibility theorem for price-adjustment mechanisms. *Proc Natl Acad Sci USA* 107(5):1854–1859.
36. Smale S (1976) A convergent process of price adjustment and global Newton methods. *J Math Econ* 3(2):107–120.
37. Kamiya K (1990) A globally stable price adjustment process. *Econometrica* 58: 1481–1485.
38. Saari DG, Simon CP (1978) Effective price mechanisms. *Econometrica* 46(5): 1097–1125.
39. van der Laan G, Talman AJJ (1987) A convergent price adjustment process. *Econ Lett* 23:119–123.
40. Herings PJ-J (2002) Universally converging adjustment processes—a unifying approach. *J Math Econ* 38:341–370.
41. Darwin C (1859) *The Origin of Species by Means of Natural Selection* (John Murray, London).
42. Graur D, Li WH (2000) *Fundamentals of Molecular Evolution* (Sinauer, Sunderland, MA).
43. Barton NH, Charlesworth B (1998) Why sex and recombination? *Science* 281(5385): 1986–1990.
44. Feldman MW, Otto SP, Christiansen FB (1996) Population genetic perspectives on the evolution of recombination. *Annu Rev Genet* 30:261–295.
45. Lewontin RC (1974) *The Genetic Basis of Evolutionary Change* (Columbia Univ Press, New York).
46. Bürger R (2000) *The Mathematical Theory of Selection, Recombination, and Mutation* (Wiley, Chichester, UK).
47. Ohta T (2002) Near-neutrality in evolution of genes and gene regulation. *Proc Natl Acad Sci USA* 99(25):16134–16137.
48. Nagylaki T (1993) The evolution of multilocus systems under weak selection. *Genetics* 134(2):627–647.
49. Chastain E, et al. (2014) Algorithms, games, and evolution. *Proc Natl Acad Sci USA* 111(29):10620–10623.
50. Livnat A (2013) Interaction-based evolution: How natural selection and nonrandom mutation work together. *Biol Direct* 8(1):24.
51. Kaznatcheev A (2013) Complexity of evolutionary equilibria in static fitness landscapes. *arXiv:*1308.5094.
52. Mézard M, Montanari A (2008) *Information, Physics and Computation* (Oxford Univ Press, Oxford).
53. Nielsen MA, Chuang IL (2000) *Quantum Computation and Quantum Information* (Cambridge Univ Press, Cambridge, UK).