Published in ACM TRANSACTIONS ON ALGORITHMS 3:2 (2007), Article 24

The NP-Completeness Column: Finding Needles in Haystacks

DAVID S. JOHNSON
AT&T Labs – Research, Florham Park, New Jersey

Abstract. This is the 26th edition of a column that covers new developments in the theory of NP-completeness. The presentation is modeled on that which M. R. Garey and I used in our book "Computers and Intractability: A Guide to the Theory of NP-Completeness," W. H. Freeman & Co., New York, 1979, hereinafter referred to as "[G&J]." Previous columns, the first 23 of which appeared in *J. Algorithms*, will be referred to by a combination of their sequence number and year of appearance, e.g., "Column 1 [1981]." Full bibliographic details on the previous columns, as well as downloadable unofficial versions of them, can be found at `http://www.research.att.com/~dsj/columns/`. This column discusses the question of whether finding an object can be computationally difficult even when we know that the object exists.

Categories and Subject Descriptors: F.1.3 [**Computation by Abstract Devices**]: Complexity Classes—*reducibility and completeness*; *relations among complexity classes*; F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General

General Terms: Algorithms, Theory

Additional Key Words and Phrases: PLS, PPAD, local search, game theory, Nash equilibrium, fixed point

1. INTRODUCTION

One way to convey the difficulty of an NP-complete problem to a nontechnical audience is to say that solving it is like trying to find "a needle in a haystack." This common expression evokes the image of searching by hand though a very tall pile of dried grass for a tiny needle buried somewhere inside. The metaphor is somewhat misleading, however, since the corresponding NP-complete problem would not concern *finding* the needle, but rather determining whether the haystack contained one.

Technically, all NP-complete problems are questions about the existence of objects. At the most fundamental level, they ask whether a fixed nondeterministic Turing machine (NDTM), given a particular input, has an accepting computation, the computation here being the object sought. Moreover, even in the more standard semantic descriptions of NP-complete problems like those in [G&J], we

Author's address: Room C239, AT&T Labs, Inc. - Research, 180 Park Avenue, Florham Park, NJ 07932, e-mail: dsj@research.att.com.

typically ask whether an object of some specified type exists – Does this graph contain a Hamiltonian circuit? Does this logical expression have a satisfying truth assignment?

On the other hand, in many applications we do want to "find the needle," that is, we want to find an object with the desired properties, not just know whether one exists. In this column I discuss the question of whether the complexity of finding the needle can differ significantly from that of telling whether one exists. More precisely, are there problems where one of the two tasks (existence testing, object finding) can be performed in polynomial time but the other cannot.

Clearly, object finding cannot be easier than existence testing. Assuming we require our object-finding algorithm to either return the desired object or correctly report that none exists, then it can also be used for existence testing. Even if we only require our algorithm to be correct on instances where the object exists, a polynomial-time object-finding algorithm will still provide a polynomial-time existence tester so long as the desired objects themselves can be recognized in polynomial time, as is typically the case. So the major question is whether object finding can be harder than existence testing.

For NP-complete problems, the answer is no. Suppose an NP-complete existence question X can be solved in polynomial time. Then all problems in NP can be solved in polynomial time, in particular, the problem X' that asks, given an instance I of X and a string y, whether there is an object of the type desired by X whose description has y as a prefix. Here, if necessary, we allow an accepting computation of the NDTM defining X to count as a description of the desired object, so that the binary-encoded description of the desired object has length bounded by a fixed polynomial p in the size $|I|$ of the input to X. Then if the desired object exists, we can construct its description with no more than $p(|I|)$ calls to the polynomial-time algorithm for solving the existence problem X'.

This argument exploits the fact that the problem is NP-complete, however. Thus there might be non-NP-complete problems in NP for which existence testing can be accomplished in polynomial time but finding a solution cannot, that is, where you can easily tell whether the haystack contains a needle, but still cannot find one quickly if it is there.

In this column, I will discuss some of the current candidates for such problems. All of these candidates are in the complexity class TFNP (total functions in NP) introduced by Megiddo and Papadimitriou [1991]. Note first that an object-finding problem, or as it is more commonly called, a "search problem," realizes a function from inputs to objects, albeit one that is not necessarily total or single-valued. To define the class TFNP, we begin with the class FNP of the search problem versions of problems in NP. Each such problem X can be viewed as consisting of a polynomial-time recognizable set I_X of instances, together with a polynomial-time recognizable relation R_X of pairs (x, y), where $x \in I_X$, y is (one of) the object(s) we wish to find for x, and the size of y is polynomially bounded in the size of x. The goal of the problem, given an instance x, is to find a y such that $(x, y) \in R_X$. A problem in FNP is in TFNP if for each $x \in I_X$ there is a y such that $(x, y) \in R_X$, that is, if the underlying function is total.

If X is a problem in NP whose existence question is solvable in polynomial time

and for which the desired objects are polynomial-time recognizable, then restricting X to those instances for which the answer to the existence question is yes yields a problem in TFNP, so the class captures the issues we wish to discuss. It is unlikely that any member of TFNP is NP-hard, since this would imply NP = co-NP [Johnson et al. 1988]). This does not, however, rule out the possibility that some members may require superpolynomial time. Let FP denote those members of TFNP which can be solved in polynomial time, that is, for which there exist polynomial-time algorithms that, given $x \in I_X$, produce a y such that $(x, y) \in R_X$. Then FP $\subseteq$ TFNP $\subseteq$ FNP, and we are looking for candidates for TFNP $-$ FP.

To date, most of the candidates belong to three subclasses of TFNP. Two of these subclasses are relatively new, while one goes back to the beginnings of the theory of NP-completeness. The old class is simply NP $\cap$ co-NP, and I will discuss the few candidates currently remaining in this class in Section 2. None of these candidates is as strong as we might wish. Each could turn out to be in P without implying that NP $\cap$ co-NP equals P. This handicap currently seems unavoidable, since NP $\cap$ co-NP seems unlikely to have complete problems (nor does the overall class TFNP). I conclude section 2 by briefly discussing why. In contrast, the two newer classes, PLS and PPAD, introduced by Johnson et al. [1988] and Papadimitriou [1994], both do have complete problems which thus provide their strongest candidates. These classes are covered in Sections 3 and 4, respectively, the latter also briefly covering several generalizations of PPAD also introduced by Papadimitriou [1994]. One inspiration for this column is the recent series of results showing that finding Nash equilibria for games is complete for PPAD, and indeed all three of the aforementioned classes contain problems about games. I will be highlighting these results and attempting to put them into perspective.

2. NP $\cap$ CO-NP

If a decision problem X is in NP $\cap$ co-NP, then the following search problem, $Answer_X$, is in TFNP: Given an instance x of X, find the answer to the question for x and a proof that the answer is correct. If X is a candidate for NP $\cap$ co-NP $-$ P, then $Answer_X$ is an equally good candidate for TFNP $-$ FP.

Unfortunately, the two most famous candidates of this sort when [G&J] was written, LINEAR PROGRAMMING and COMPOSITE NUMBER, are now both known to be in P. The first was known to be in NP $\cap$ co-NP because of linear programming duality, and was shown to polynomial-time solvable by Khachiyan [1979] very shortly after the appearance of [G&J]. The second was in NP because any nontrivial factor of n constitutes a proof of compositeness, and in co-NP because there are also simple proofs of primality [Pratt 1975]. Its candidacy lasted much longer, but membership in P was eventually proven by Agrawal et al. [2004] (see Column 24 [2005]). The related problem of factoring composite numbers is still a candidate, however, and can be viewed as $Answer_X$ for the following decision problem.

SMALL FACTOR

INSTANCE: Integers n and k, $k \leq n$.

QUESTION: Does n have a factor lying strictly between 1 and k?

The (unique) prime decomposition of n will constitute a proof of the answer

to this question, whether yes or no, and this proof is polynomial-time checkable because primality can now be verified in polynomial time. Note that this means that SMALL FACTOR is not only in NP ∩ co-NP but also in UP ∩ co-UP, where UP is the set of decision problems that can be solved by Unambiguous Turing Machines (UTMs) in polynomial time, and a UTM is an NDTM that has either 1 or 0 accepting computations, as discussed in Column 15 [1985]. The UTMs in question would simply guess the prime decomposition of n in some canonical form, and then run a deterministic algorithm to validate the decomposition and compare the smallest factor to k. The class UP ∩ co-UP is interesting in another context (cryptographic), since worst-case one-way permutations exist if and only if P ≠ UP ∩ co-UP [Homan and Thakur 2003].

Note that SMALL FACTOR is polynomial-time equivalent to the problem of finding the unique prime decomposition, as the latter can be determined using only a polynomial number of calls to a subroutine for SMALL FACTOR: Start with $i = 1$ and initial divisor $m = 1$ of n. While $n/m > 1$, perform $O(\log n)$ calls to the subroutine in a binary search procedure to find the smallest prime factor p_i of n/m, set $m = mp_i$, set $i = i+1$, and continue. We will be done after performing at most $\log n$ passes through the `while` loop, yielding a total of $O(\log^2 n)$ subroutine calls, which is polynomial in the size of n.

Currently, the best algorithm known for factoring is still far from polynomial, having a running time that by plausibel arguments is $O(2^{cn^{1/3}\log^{2/3} n})$ for n-digit numbers and some constant c [Lenstra and Lenstra 1993; Pomerance 1996]. The assumption that factoring is hard underpins the presumed security of several cryptosystems, such as RSA. Other popular cryptosystems are based on the assumption that a second number-theoretic problem is hard to solve: the discrete logarithm problem. Here we are given a prime p, a generator a for the group $\mathbf{Z}_p$ (integers mod p), and an integer x, $0 \le x \le p-1$, and wish to find the *discrete logarithm* of x with respect to p and a, which is the k such that $x \equiv a^k \bmod p$. By arguments similar to those for factoring, this problem can be seen to be equivalent to a decision problem in NP ∩ co-NP, in this case the problem SMALL DISCRETE LOG, analogously defined. Once again, this latter problem is in fact in UP ∩ co-UP.

Although SMALL FACTOR and SMALL DISCRETE LOG are the most famous surviving candidates for membership in NP ∩ co-NP − P, several additional candidates have surfaced since the publication of [G&J]. All are game-theoretic and all, like the previous two, are in UP ∩ co-UP. The first was identified by Condon [1992].

SIMPLE STOCHASTIC GAME

INSTANCE: Directed graph $G = (V, A)$ with maximum outdegree 2 and minimum outdegree 1, specified *start* and *end* vertices v_0 and v_1, and a partition of V into three sets V_{min}, V_{max}, and $V_{average}$.

QUESTION: Does player P_1 have a strategy that will guarantee she wins with probability exceeding 1/2, no matter what strategy is used by player P_2, in the following 2-player game? We start with a token at v_0. Suppose now that the token is at vertex v. If $v = v_1$, the game is over. Otherwise, if $v \in V_{min}$ then player P_1 chooses a vertex u such that $(v, u) \in A$ and moves the token to u. If $v \in V_{max}$, then player P_2 chooses a vertex u such that $(v, u) \in A$ and moves the token to u.

If $v \in V_{average}$ and $outdegree(v) = 1$, the token is moved to the unique successor vertex of v. Otherwise, $outdegree(v) = 2$ and the successor vertex to which the token is moved is chosen randomly, with equal probabilities for each choice. Player P_1 wins if the token ever moves to v_1. Otherwise, player P_2 wins.

Condon [1992] shows that we can in polynomial time transform any game G meeting the definition to a specially structured game G', for which the following properties hold: (1) Player P_1 has a strategy that wins with probability exceeding $1/2$ for G (no matter what strategy P_2 uses) if and only if P_1 has such a strategy for G'. (2) There is a unique valuation function p that assigns to each vertex v in game G' the rational probability that P_1 will win in game G', assuming the token is at vertex v and P_1 plays her optimal strategy against best play by P_2. (3) The correctness of the valuation function can be verified in polynomial time simply by checking that certain local recurrence relations hold. Thus the NDTMs (UTMs) that solve the problem and its complement simply construct G', guess the valuation in a canonical form, and then deterministically verify its correctness and compare the value $p(v_0)$ to $1/2$. Interestingly, the problem is in P if any one of the three types of vertices (*min*, *max*, and *average*) is not present [Condon 1992].

Our second game-theoretic example comes in several variants, and we will discuss a version covered by Zwick and Paterson [1996], although the basic result was first observed by Karzanov and Lebedev [1993].

MEAN PAYOFF GAME

INSTANCE: Bipartite directed graph $G = (V_1, V_2, A)$, all of whose vertices have outdegree at least 1, together with an integer weight function $w : A \to \mathbf{Z}$, a specified vertex $v_0 \in V_1$, and an integer C.

QUESTION: Does player P_1 have a winning strategy in the following (nonstochastic) 2-player game? Initially player P_1 picks an arc $a = (v_0, v_1) \in A$, thus starting a path. Thereafter, players alternate picking arcs, where if the current path ends with vertex v, then the next player must pick some arc $(v, u) \in A$. The game ends when the arc chosen completes a cycle, that is, when the endpoint u is a vertex that already occurred earlier in the path. P_1 wins if the average arc length in the cycle thus formed is C or less.

This game is equivalent (somewhat surprisingly) to the infinite game in which play never stops, and player P_1 wants to guarantee $\limsup_{n\to\infty}(1/n)\sum_{i=1}^{n} w(a_i) \leq C$, where a_i is the ith arc chosen [Ehrenfeucht and Mycielski 1979]. That the problem is in NP ∩ co-NP follows from a valuation result similar to that for SIMPLE STOCHASTIC GAME and proved by Gurvich et al. [1988]. That it is in UP ∩ co-UP follows most clearly from the fact that there is a polynomial transformation from MEAN PAYOFF GAME to SIMPLE STOCHASTIC GAME [Zwick and Paterson 1996]. The first part of this transformation converts instances of the former to instances of a generalized version of the latter in which the graph can have arbitrarily large outdegrees, and for each vertex in $V_{average}$ there is a rational probability distribution over its out-arcs, according to which the successor vertex is chosen. Instances of this generalized problem are then transformed to instances of SIMPLE STOCHASTIC GAME itself, using a modification of a construction from Condon [1992].

As a final example, consider the PARITY GAME, introduced by Emerson and Jutla [1991] in the context of model checking. This is like the infinite version of MEAN PAYOFF GAME, except that now player P_1 wins if the largest weight on an infinitely recurring edge is even. The question of whether P_1 has a winning strategy is reducible to MEAN PAYOFF GAME and hence is in UP$\cap$co-UP [Puri 1995; Jurdziński 1998]. Thus in a sense it is no harder than the other two problems. Nevertheless, no polynomial-time algorithm is known for it. The best deterministic algorithm currently known for it is due to Jurdziński et al. [2006] and runs in time $n^{O(\sqrt{n})}$, where n is the number of vertices in the graph.

At present, the preceding problems (and variants thereof) are the only candidates I know for (nontrivial) membership in NP $\cap$ co-NP $-$ P. It would be interesting to identify other candidates, and in particular to find candidates that do not also appear to be in UP $\cap$ co-UP. It would also be nice to have a candidate for which the evidence of hardness is more than simply "we haven't yet figured out a way to solve this particular problem in polynomial time." Ideally, we would like a candidate that could only be in P if NP $\cap$ co-NP = P, the kind of evidence of hardness one typically gets from completeness results.

Unfortunately, it is unlikely that NP $\cap$ co-NP has complete problems. For instance, there is an oracle A such that there are no complete problems (under polynomial transformations) for NP^A $\cap$ co-NP^A [Sipser 1982], which implies that no argument that relativizes can be used to prove a completeness result for NP $\cap$ co-NP. As a perhaps more intuitive argument for why it might be hard to prove a completeness result for a class such as NP $\cap$ co-NP, consider the following.

For any class, we would seem to need an analogue of Cook's Theorem. What is involved in the proof of such a theorem? At a fundamental level, all such proofs currently known are in a sense themselves polynomial transformations. Given a description of a problem X in the class, they produce a polynomial-time procedure P_X for solving X given a black-box subroutine for the target (supposedly complete) problem Y. (Depending on the type of reduction we are using, be it many-one, Turing, or something else, there may be restrictions on how the black-box subroutine may be used.) For the proof to be verifiable, the construction of P_X, given the description of X, must itself be efficiently computable – in the proof of Cook's theorem itself, the construction in fact takes linear time. Presumably the construction must also exploit the properties that define the class, so it is not too much of a leap to expect that the set of problem descriptions for the class should itself be recognizable in polynomial time.

Thus, in the current state-of-the-art for completeness proofs, it is essential that there be a polynomial-time recognizable representation for members of the class. For NP, such a representation is provided by a polynomial p and an NDTM with a standard built-in step-counting procedure that can be verified to cause the NDTM to halt after $p(|x|)$ steps. Similar representations exist for PSPACE, #P, and other classes for which complete problems are known to exist. Note that this typically allows us to obtain a *generic* complete problem for the class, in which the instances are pairs $(D(X), x)$, where $D(X)$ is the description of a problem X in the class and x is an instance of X, and the answer for $(D(X), x)$ is the answer for instance x in problem X. This will be a *problem* as normally defined, since its instances

are polynomial-time recognizable, and will be complete because all members in the class reduce to it, essentially by instantiation. Cook's Theorem can in fact be viewed as a reduction from the generic problem for NP to SATISFIABILITY.

Applying this reasoning to NP ∩ co-NP, we are unlikely to prove an analogue of Cook's Theorem unless we can provide a polynomial-time recognizable characterization of all the problems in the class. Unfortunately, the natural one doesn't work. Given polynomial-time recognizable descriptions of two polynomial-time NDTMs, it is in fact *undecidable* to determine whether they recognize complementary languages. This is because we can embed the halting problem in this question. Suppose we wish to determine whether Turing machine M halts on input $x \in \{0,1\}^*$. We construct two clocked linear-time Turing machines M_1 and M_2. On input y, M_1 simply reads the input and always accepts. M_2 always rejects unless $|y| \geq |x|$ and M accepts x in $|y|$ or fewer steps. Thus M_1 and M_2 will accept complementary languages if and only if M does not halt on input x. Hence if the problem of whether two clocked polynomial-time Turing machines accept complementary languages is decidable, so is the halting problem, contradicting the undecidability of the latter problem. Thus problems in NP ∩ co-NP are not polynomial-time recognizable given the standard representation for them. There might be some other representation of NP ∩ co-NP that *is* polynomial-time recognizable, but finding one would be a major new result.

A similar argument applies to the overall class TFNP. Given a polynomial-time recognizable description of a polynomial-time NDTM, it is undecidable to determine whether the NDTM accepts all inputs. Hence this class too is unlikely to have complete problems. Fortunately, it contains interesting and nontrivial subclasses that do have complete problems. We cover these in the next two sections.

3. PLS

In this section we discuss the class PLS, introduced by Johnson et al. [1988], where "PLS" stands for "polynomial-time local search." In a local search algorithm for a combinatorial optimization problem, one exploits a neighborhood structure on the feasible solutions to the problem, where two solutions are neighbors if one can get from one to the other by a simple alteration of the first. As an example, consider the MAX CUT problem in which we are given a graph and wish to partition its vertices into two sets so as to maximize the number of edges with endpoints in different sets. Here a feasible solution is any partition of the vertices into two sets, and a simple local search algorithm would use the *flip* neighborhood structure where two solutions are neighbors if they differ in the location of a single vertex. The local search algorithm would use some heuristic to generate an initial *current solution*. It would then repeatedly update the current solution by replacing it with a neighbor of better cost, until it reached a partition such that no movement of a single vertex could improve it, that is, one that was "locally optimal" with respect to the neighborhood structure. Note that so long as the solution space is finite, such local optima must exist and the algorithm is guaranteed to find one.

The world of optimization is full of local search algorithms, including for instance the well-known "2-Opt" heuristic for the traveling salesman problem (TSP), where the instances are complete weighted graphs whose vertices are called *cities*, the

feasible solutions are Hamiltonian cycles (*tours*), and two tours are neighbors if one can be obtained from the other by deleting two edges and adding two new ones.

Local search algorithms can be quite effective in practice, although they do not necessarily provide good worst-case guarantees. For instance, although 2-Opt tends to find very good tours on real-world TSP instances [Johnson and McGeoch 1997], there are instances where all but one of the locally optimal tours are worse than optimal by an exponential factor [Papadimitriou and Steiglitz 1978], and even for instances obeying the triangle inequality there are instances with local optima that are off by an $\Omega(\sqrt{n})$ factor [Korupolu et al. 2000], where n is the number of cities. On the other hand, for some problems, local search algorithms with constant-factor worst-case guarantees do exist. This is the case, for example, for the facility location problems covered in Arya et al. [2004]. Also, the simplex algorithm for LINEAR PROGRAMMING can, with appropriate tie-breaking rules, be viewed as a local search algorithm that is guaranteed to find an optimal solution.

Here, however, we are not concerned with the quality of the local optima, but with the problem of finding them. Although the above local search algorithms tend to find local optima quickly in practice, there is no guarantee that they will always do so. Indeed, for many variants on the simplex algorithm there are sequences of instances for which their running times grow exponentially, as first observed by Klee and Minty [1972]. Similarly, there are instances for which 2-Opt can take exponential time [Lueker 1976], even in the case of points in the plane under the Euclidean metric [Englert et al. 2007]. Such results, however, refer only to the operation of the particular local search *algorithms*, not to the basic problem of finding a solution that is locally optimal under the given neighborhood structure. There might well be some more efficient way to construct a locally optimal solution than by simply applying the local search algorithm itself. This is for instance the case with LINEAR PROGRAMMING, where the ellipsoid algorithm can in polynomial time find an optimal solution (which is of necessity locally optimal under the simplex neighborhood structure), even though no algorithm that performs local search within the simplex neighborhood structure is known to obey such a time bound. There are nonetheless many local search neighborhoods, including that of 2-Opt, for which we know of no way, direct or indirect, to find local optima in polynomial time.

These problems are consequently a promising potential source of additional candidates for problems in TFNP − FP. Locally optimal solutions are guaranteed to exist as long as the solution space is finite. Thus membership in TFNP follows from the fact that the local optimality of a solution can be verified in polynomial time. Most of these problems fall into the class PLS, which was defined so that its members can be recognized in polynomial time, thus yielding the possibility of complete problems.

Formally, a problem X in PLS can be specified by the following:

(1) A polynomial bound (in terms of the size of an instance x of X) on the size of a solution y for x, and the identification of X as either a minimization or a maximization problem.

(2) A polynomial-time algorithm A_X that, given a string x, determines whether x is an instance of X and if so, outputs an (initial) solution y for x.

(3) A polynomial-time algorithm C_X that, given an instance x of X and a string

y, determines whether y is a solution for x and if so, outputs an integer *cost* value $c(x, y)$.

(4) A polynomial-time algorithm F_X that, given (x, y) where x is an instance of X and y is a solution for x, either reports "locally optimal" or returns a solution $f(x, y) = y'$ for x such that $c(x, y') < c(x, y)$ if X is a minimization problem or $c(x, y') > c(x, y)$ if X is a maximization problem,

The reader can verify that it is possible to combine these elements into a standard, polynomial-time recognizable method for describing problems in the class. Note that this definition, from Johnson et al. [1988], does not explicitly specify the neighborhood structure. The neighborhood of y could simply be the singleton set $\{f(x, y)\}$ or could even be some exponential size set from which the best solution can be found in polynomial time. An alternate, more restrictive definition, given by Papadimitriou [1994], replaces F_X by a polynomial-time algorithm that produces a list of all the neighbors of y, in which case the neighborhoods must be of polynomial-size. The algorithm for finding a better neighbor if one exists is then just the trivial one that computes the cost of each neighbor and outputs one that is strictly better than y if any such exists. In either case, given this information about X, the problem of finding a locally optimal solution for an instance of X is clearly in TFNP – the relevant NDTM would simply guess a locally optimal solution, use algorithm C_X to verify that it is a valid solution for x, and then use algorithm F_X to confirm that it has no better neighbor.

To prove completeness for PLS, we need an augmented form of reduction that applies to search problems in TFNP. Generalizing the definition of *PLS reduction* from Johnson et al. [1988], let us say that a *TFNP reduction* from problem X to problem Y consists of the following two polynomial-time computable functions:

(1) A function f that takes each instance x of X to an instance $f(x)$ of Y.

(2) A function g that, for each instance x of X and answer y for instance $f(x)$ in problem Y, yields an answer $g(x, y)$ for x in problem X, where *answers* are the objects sought in the search problems.

It is easy to see that, given such a reduction from PLS problem X to PLS problem Y, where the answers are locally optimal solutions, the existence of a polynomial-time algorithm for finding locally optimal solutions in Y implies the existence of a polynomial-time algorithm for finding locally optimal solutions in X. Thus if a problem is complete for PLS under such reductions, it can be in P if and only if PLS = P. In Johnson et al. [1988], a PLS-specific variant on Cook's Theorem was used to identify the following minimization problem as the "first" (non-generic) PLS-complete problem.

CIRCUIT FLIP

INSTANCE: An acyclic Boolean circuit f made up of *and*, *or*, and *not* gates and having m inputs and n outputs.

SOLUTIONS: Length-m binary vectors representing possible inputs to the circuit, with the *initial solution* being the all-1 vector.

COST: If x is a solution and $y = (y_1, y_2, \ldots, y_n)$ is the length-n binary output vector resulting when f is applied to x, then the cost $c(f, x) = \sum_{i=1}^{n} 2^i y_i$.

NEIGHBORS: The neighbors of a solution x are all those length-m binary vectors that differ from x in exactly one bit, where if there is a neighbor of x with lower cost, we always return the one with lowest cost, ties broken lexicographically.

It is not difficult to see that, given this definition, the polynomial-time algorithms A, C, and F required by the definition of PLS can be constructed. Although this is basically a made-up problem, it has proved to be a successful starting point for proving PLS-completeness of a variety of local-optimum-finding problems derived from real-world local search algorithms.

For example, WEIGHTED MAX CUT, in which edges have weights and we wish to maximize the sum of the weights of edges in the cut, is PLS-complete under the previously mentioned *flip* neighborhood structure [Schäffer and Yannakakis 1991]. GRAPH PARTITIONING, the variant on WEIGHTED MAX CUT in which we require the two sets on opposite sides of the cut to be of equal size and wish to *minimize* the weight of the cut, is PLS-complete under the *swap* neighborhood structure where two cuts are neighbors if one is obtained from the other by picking a pair of vertices on opposite sides of the cut and swapping their locations [Schäffer and Yannakakis 1991]. WEIGHTED MAX-2SAT is PLS-complete under the *flip* neighborhood structure in which two truth assignments are neighbors if they differ in value of a single variable [Schäffer and Yannakakis 1991]. It is not known whether the TSP is PLS-complete under the 2-Opt neighborhood structure, although Krentel [1989] has shown that for some $k < 10,000$, it is PLS-complete under the k-Opt neighborhood structure where two neighbors differ by up to k edges. In addition, the TSP is PLS-complete under a neighborhood structure derived from a variant of the famous Lin-Kernighan algorithm [Lin and Kernighan 1973] in which neighbors are determined by an algorithmically-derived, variable-length sequence of 2- and 3-Opt moves [Papadimitriou 1992]. Similarly, GRAPH PARTITIONING is PLS-complete under the neighborhood structure derived from the Kernighan-Lin algorithm [Kernighan and Lin 1970] in which neighbors are determined by an algorithmically-derived, variable-length sequence of swaps [Johnson et al. 1988].

In the preceding descriptions we do not specify how the initial solution is specified or how ties are broken when more than one better neighbor exists. The relevant references provide the details (typically any tie-breaking function can be used in the neighbor-finding algorithm).

Not all PLS-complete problems arise in the context of local search algorithms, where we want locally-optimal solutions simply because we hope they will be near-optimal. In some cases the desired property is simply local optimality itself, as the following two problems illustrate.

STABLE CONFIGURATION FOR A NEURAL NET

INSTANCE: An undirected graph $G = (V, E)$, together with an integral weight $w(e)$ for each edge $e \in E$, and an integral threshold $t(v)$ for each vertex $v \in V$.

GOAL: Find a stable configuration, where a *configuration* is an assignment of a value $s(v) \in \{-1, +1\}$ to each $v \in V$, a vertex v is *stable* in a configuration if either (a) $s(v) = 1$ and $\sum_{\{u,v\} \in E} s(u)w(\{u, v\}) \geq t(v)$, or (b) $s(v) = -1$ and $\sum_{\{u,v\} \in E} s(u)w(\{u, v\}) \leq t(v)$, and a configuration is stable if all its vertices are stable.

Hopfield [1982] showed that this problem is in TFNP by introducing the potential function $\sum_{\{u,v\}\in E} w(\{u,f\})s(u)s(v) - \sum_{v\in V} t(v)s(v)$ and showing that if a vertex v is unstable, then flipping the value of $s(v)$ will increase the value of the potential function, while if a vertex is stable, flipping its value will decrease the potential function. Since the set of configurations is finite, this means that local optima for the potential function must exist under the flip neighborhood, and there is a one-to-one correspondence between stable configurations and local optima for the maximization problem based on this potential function and the *flip* neighborhood structure. This problem was proved PLS-complete by Schäffer and Yannakakis [1991].

PURE NASH EQUILIBRIUM IN A CONGESTION GAME

INSTANCE: A finite set R of *resources*, a set $p_1, p_2, \ldots, p_n$ of players, for each player p_i a finite set S_i of *strategies*, where each strategy is a subset of R, and for each resource $r \in R$, a polynomial-time computable nondecreasing *congestion function* $f_r : \{0, 1, \ldots, n\} \to \mathbf{Z}$.

GOAL: Consider the following game: Each player p_i chooses one of his strategies $s_i \in S_i$. For each resource $r \in R$, let r's *congestion* n_r be the number of players whose chosen strategy contains r. Then the *penalty* for player p_i is $\sum_{r\in s_i} f_r(n_r)$, and the goal of each player is to minimize his penalty. A *pure Nash equilibrium* for this game is an assignment of strategies to players such that no individual player can reduce his penalty by changing his strategy.

That the problem of finding such an equilibrium for a congestion game is in TFNP follows from a result of Rosenthal [1973], again involving a potential function argument (see also Fabrikant et al. [2004], Vöcking [2006]). The desired equilibria are in one-to-one correspondence with the local optima for the potential function under the "change-one-player's-strategy" neighborhood structure, and so the problem can be viewed as belonging to PLS. It is PLS-complete, even in the restricted case of *symmetric* games, where all the strategy sets S_i are identical. This was proved by Fabrikant et al. [2004], who also proved PLS-completeness for the problem of finding a pure Nash equilibrium in the much more structured NETWORK CONGESTION GAME. In such a game there is an underlying graph $G = (V, E)$ (the *transport network*) and the edges of the graph are the resources. Associated with each player p_i is a pair of vertices (u_i, v_i), with the player wishing to send one unit of traffic from u_i to v_i. The strategy set for p_i then consists of all paths in the graph from u_i to v_i (where each path is viewed as a set of edges). The congestion function for an edge then represents the *delay* on that edge as a function of the number of players that use it in their chosen path. In contrast to the general problem, here the restriction to symmetric games makes the problem polynomial-time solvable (via network flow techniques) [Fabrikant et al. 2004].

For additional PLS-completeness results about congestion games, see the above-mentioned references. For an excellent early survey of PLS-completeness that discusses many other examples of PLS-complete problems, see Yannakakis [1997].

Let me conclude this section by returning to a question about local search algorithms themselves. Given a problem X in PLS, let us say the *standard algorithm* is the one that starts with the initial solution produced by algorithm A_X, and

thereafter always moves to the solution generated by F_X until that algorithm reports that there are no neighbors with better costs. We know that for problems in PLS, this algorithm will always (eventually) find a local optimum, although we have already seen examples of polynomial-time solvable problems in PLS where the standard algorithm can take exponential time (as with the simplex algorithm for LINEAR PROGRAMMING). What can we say about the standard algorithms for PLS-complete problems? It can be shown that for CIRCUIT FLIP the standard algorithm does take worst-case exponential time. Moreover, by requiring a bit more of our completeness-proving reductions (i.e., by using the *tight* PLS-reductions of Papadimitriou et al. [1990] and Yannakakis [1997]), this result can be extended to most other known PLS-complete problems. Given these results, one might well ask if there could possibly be an easier way to find the solution output by the standard algorithm than simply to run the algorithm itself. This seems unlikely. Finding the locally optimal solution generated by the standard algorithm is PSPACE-hard for CIRCUIT FLIP, a result that again extends to most of the other PLS-complete problems via tight PLS-reductions [Papadimitriou et al. 1990; Yannakakis 1997]. Thus from a complexity-theoretic standpoint, it seems to be much harder than simply finding *any* locally optimal solution.

I should note that all these results depend on the fact that we use binary representations for the weights, thresholds, and other numerical parameters in the underlying problems. The unweighted (or unary-weighted) versions of these problems are all in P because in these cases the objective/potential function is bounded by a polynomial and the standard algorithm can only take a polynomial number of steps. The problems are still logspace complete for P, however, and so are unlikely to be solvable in polylogarithmic parallel time using only a polynomial number of processors [Papadimitriou et al. 1990; Schäffer and Yannakakis 1991].

4. PPAD

In the previous section we studied a class of problems that are in TFNP essentially because every finite set of numbers has a minimal element. Alternatively, we could say they are in TNFP because every directed acyclic graph has a sink. The vertices of the graph for problem X are the solutions for the given instance x and there is an arc from solution y to solution y' if the latter is the output when F_X is applied to the pair (x, y). The graph is guaranteed to be acyclic since we require that $c(x, y')$ be strictly better than $c(x, y)$. Moreover, note that the graph is highly restricted, since all vertex outdegrees are either 0 or 1.

The definition of the class PPAD relies on an equally elementary graph-theoretic observation for the proof that its members are in TFNP. It was originally introduced under the name PDLF by Papadimitriou [1990], but was renamed PPAD (for "Polynomial Parity Argument, Directed version") in the journal version of that paper [PaPADimitriou 1994]. To define PPAD, we start with a polynomial-time recognizable base class PPAD_0 and then enlarge it to contain all problems in TFNP that are TFNP-reducible to a problem in PPAD_0. A problem X is in the class PPAD_0 if we can specify the following.

(1) A polynomial bound (in terms of the size of an instance x of X) on the size of a solution y for x.

(2) A polynomial-time algorithm A_X that, given a string x, determines whether x is an instance of X and if so produces a specified initial *source* solution y_0.

(3) A polynomial-time algorithm P_X that, given an instance x of X and a string y, determines whether y is a solution for x and if so outputs a string $pred(x, y) = y'$, with $pred(x, y_0) = y_0$.

(4) A polynomial-time algorithm S_X that, given an instance x of X and a string y, determines whether y is a solution for x and if so outputs a string $succ(x, y) = y'$, with $succ(x, y_0) \neq y_0$ and $pred(x, succ(x, y_0)) = y_0$.

The goal for a problem described in this way is to find a solution y other than y_0 that has indegree(y) + outdegree(y) = 1 in the directed graph whose vertices are solutions and whose arcs are those vertex pairs (u, v) where both $u = pred(v)$ and $v = succ(u)$. Since y_0 is guaranteed to have indegree 0 and outdegree 1 (is a *source*) and the graph is finite, such a solution y must exist. Indeed, a vertex with indegree = 1 and outdegree = 0 (a *sink*) must exist, although sources other than y_0 are acceptable answers as well. Thus the problems so defined are all in TFNP.

It is easy to see that descriptions of the previous form can be cast in polynomial-time recognizable form, and hence PPAD_0 can potentially have complete problems. Papadimitriou [1994], in addition to introducing the class PPAD, also shows that such problems do exist. One can start with the generic complete problem for PPAD_0, called END OF THE LINE in Daskalakis et al. [2006], where we are given a description of a problem X in the class and an instance x of X, and are asked for a sink or a source other than y_0 in the graph implicitly defined by X for x. Note that, by the transitivity of TFNP reductions, END OF THE LINE is also complete for all of PPAD. It and any other complete problems for PPAD will constitute plausible candidates for TFNP − FP, since there are many problems in PPAD for which we know of no polynomial-time algorithms.

The first (non-generic) problem shown to be PPAD-complete was a 3-dimensional version of the Sperner problem, which can be defined in general as follows [Grigni 2001]. A *simplex* in d-dimensional space (a *d-simplex*) is the convex closure of a set of $d+1$ affinely independent points (*vertices*). Affine independence implies that all $d + 1$ points are extreme points of the simplex. A *face* of the simplex is the convex closure of any proper subset of these extreme points, with the face being a *facet* if the subset has d members. When $d = 2$ the simplices are triangles whose edges are the facets. When $d = 3$ the simplices are tetrahedrons whose planar faces are the facets. A d-triangulation in $\mathbf{R}^d$ consists of an *external* d-simplex which is itself subdivided into multiple d-simplices, where any two of these simplices (one of which might be the external one) are either disjoint or share a common face. If that common face is a facet, we shall say the simplices are *neighbors*. A *proper coloring* of a d-triangulation assigns each vertex of the triangulation an integer from $\{0, 1, \ldots, d\}$, with the external simplex being *panchromatic* (receiving all $d+1$ colors).

Sperner's Lemma [Sperner 1928] states that in a proper coloring, one of the interior simplices must also be panchromatic. In general, it follows from an undirected version of the graph property we used to show PPAD ⊆ TFNP. Consider the graph in which the simplices of our triangulation are the vertices, and two simplices are adjacent if they share a common facet that has all the colors except 0. It is not

difficult to see that a panchromatic simplex has degree 1 in this graph, while all other simplices either have degree 0 or 2. Since any graph must have an even number of odd-degree vertices (the "parity" argument abbreviated by the second P in PPAD), we conclude that the desired internal simplex must exist. So far the graph is undirected, but it is easy to see how it can be made directed in the case of $d \in \{2, 3\}$. Let the *prime* facet of the external simplex be the one that omits the vertex colored 0. When $d = 2$ the facet is an edge and has two possible orientations, 1-2 (1 to the left) and 2-1 (2 to the left). When $d = 3$ the facet is a triangle and again has two possible orientations, the clockwise and counterclockwise order for 1-2-3. The *positive orientation* is the orientation of the prime facet that is seen when that facet is viewed from the interior of the neighbor of the external simplex. The edge between two adjacent simplices s and s' is directed toward s' if the common facet, when viewed from the interior of s', has the positive orientation. It can be seen that this scheme insures that no simplex has in- or outdegree exceeding 1. Thus the problem of finding the second panchromatic simplex will be in PPAD so long as we can produce the required polynomial-time algorithms for specifying the triangulation.

This is done in the 3-dimensional case by basing the triangulation on an $n \times n \times n$ rectilinear grid with vertex coordinates in the set $\{0, 1, \ldots, n\}$. This grid can be viewed as made up of n^3 unit cubes, each partitioned into five tetrahedrons by a standard construction. The vertices $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$ then correspond topologically to the vertices of the external tetrahedron, which can be rendered external by stretching these vertices away from the grid until all the other vertices of the grid are on the inside. The triangulation is then completed by adding edges from these four vertices to the vertices on the faces of the grid. Sperner's Lemma can then be applied to show that the following problem is in PPAD.

3D SPERNER

INSTANCE: Integer n written in binary and a polynomial-time algorithm for computing a function $f : \{0, 1, \ldots, n\}^3 \rightarrow \{0, 1, 2, 3\}$ such that (i) $f(0, 0, 0) = 0$, $f(1, 0, 0) = 1$, $f(0, 1, 0) = 2$, and $f(0, 0, 1) = 3$, (ii) for each of the six external faces of the corresponding $n \times n \times n$ grid, the vertices in that face collectively receive at most three distinct colors, and (iii) the only faces whose vertices can receive color 0 are the ones containing $(0, 0, 0)$.

GOAL: Find a panchromatic unit cube C in the grid.

Note that the restrictions on the colorings of the vertices in the external faces of grid insure that, when we add edges from the four chosen vertices to the vertices on the faces of the grid, we can do so in such a way that none of the tetrahedrons created is panchromatic. Thus the only panchromatic tetrahedron not contained in one of the unit squares is the one consisting of the four vertices $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$, and consequently the second panchromatic tetrahedron implied by Sperner's lemma must lie within one of the unit squares. So the problem is in PPAD.

3D SPERNER was proved PPAD-complete in Papadimitriou [1994], along with a variety of other problems, including discrete problems based on the fixed point theorems of Brouwer [1910] and Kakutani [1941], a variant on the LINEAR COMPLE-

MENTARITY PROBLEM of mathematical programming, and an economics problem related to equilibrium pricing. The paper also proved that several other problems in TFNP and not known to be in FP were in PPAD and hence candidates for PPAD-completeness. One was the 2-dimensional version of 3D SPERNER, essentially obtained by dropping the third dimension and the fourth color. This was recently proved PPAD-complete by Chen and Deng [2006a]. The other was the problem of computing a (mixed) Nash equilibrium in a 2-player game.

To understand this latter problem, let us return to the problem of finding *pure* Nash equilibria, discussed in the previous section in the context of congestion games. In a general game we are given a set $\{p_1, p_2, \ldots, p_n\}$ of players and, for each player p_i, a finite set S_i of *strategies* and a function $f_i : S_1 \times S_2 \times \cdots \times S_n \to \mathbf{Q}$, where $f_i(s_1, s_2, \ldots, s_n)$ gives the *payoff* to player p_i if s_j is the strategy picked by player p_j, $1 \leq j \leq n$. A *pure* Nash equilibrium for such a game is a choice of a single strategy for each player such that no player can improve his payoff by changing strategies. For congestion games, pure equilibria are guaranteed to exist, but in general they are not, and it is NP-hard to tell whether one exists [Gottlob et al. 2003; Álvarez et al. 2005]. Now, technically speaking, if the game is represented as before, with instances including a full listing of the values of the payoff functions for all inputs, then an exhaustive search for a pure Nash equilibrium takes only polynomial time in the input size. However, in many applications such as congestion games, it is hopelessly inefficient to use such an explicit representation since the payoff functions, and even the strategy sets, may have compact (implicit) descriptions and, with such representations, NP-completeness may be a real possibility.

Consider, for example, *graphical* games, in which the players correspond to the vertices of a graph, and their payoffs depend only on their own strategy and the strategies chosen by their immediate neighbors in the graph. If the maximum vertex degree is bounded by a constant, then descriptions of the payoff functions for each vertex can be polynomially bounded in the maximum number of strategies per player. Gottlob et al. [2003] show that telling whether such a game has a pure Nash equilibrium is NP-complete even if the maximum vertex degree is three and no player has more than three strategies.

The classical notion of a *Nash equilibrium* is not, however, that of a pure equilibrium but instead of a *mixed* equilibrium, and the classical result of Nash [1951] is that such equilibria, which I will define next, always exist. In a mixed equilibrium, the players do not choose a single strategy, but instead specify a probability distribution over their strategies, and choose their strategies according to their distributions. Thus we are now concerned with expected payoffs, rather than fixed payoffs. In a mixed Nash equilibrium, no player can improve his expected payoff by changing his distribution. Nash proved that such equilibria exist by means of the Kakutani fixed point theorem. Note that mixed equilibria are much more complicated objects than pure equilibria. In particular, even for 2-player games in which each player has n strategies, the number of possible distributions to consider need not be polynomially bounded in n, whereas for pure Nash equilibria there would be only n^2 possibilities.

Thus finding (mixed) Nash equilibria might be difficult even for a fixed number of players and an explicit presentation of the payoff matrix. The 2-player version

of this problem (2-PLAYER NASH) was highlighted in Papadimitriou [1994], which showed that it was in fact in PPAD. That it is even in TFNP requires more than Nash's result. In addition we exploit the following two observations: (a) The distributions in a 2-player Nash equilibrium are all rational with representations that are polynomial in the size of the payoff matrix [Papadimitriou 1994], and (b) one can verify that a given set of distributions is a Nash equilibrium simply by showing that no player can improve his payoff by changing his strategy to a single pure strategy. That the problem is in PPAD then follows from the observation that a deterministic algorithm due to Lemke [1965], which works by pivoting from one distribution set to another, is guaranteed to find a Nash equilibrium (although it may take exponential time). The algorithm's pivot step induces the directed graph required by the definition of PPAD [Papadimitriou 1994].

Things get a bit more complicated if there are more than two players. As soon as there are even three players, we are no longer guaranteed that the probabilities in a Nash equilibrium are rational [Nash 1951]. Goldberg and Papadimitriou [2006] addressed this issue by adding an additional accuracy parameter A, consisting of a string of 0's, to each instance. We can then ask for a set of distributions such that no player can improve his expected payoff by more than $\epsilon = 1/2^{|A|}$. Proving such a problem hard will in a sense be an inapproximability result, but it will be a very weak one, since it would not even rule out the possibility of a fully polynomial-time approximation scheme (FPTAS). Furthermore, for the case of two players, the problem with the accuracy parameter is equivalent to the one without one, since equilibria have rational probabilities of polynomial-bounded length. Adding the accuracy parameter allows us to say that r-PLAYER NASH is in TFNP for $r > 2$, but not that it is in PPAD. For that we also need some new ideas, since the previously mentioned proof using Lemke's algorithm only applies when $r = 2$. Fortunately, a more general proof exploiting Sperner's Lemma does exist [Daskalakis et al. 2007].

The problems r-NASH for fixed $r \geq 2$ appear to be hard. The best algorithms currently known for them are those of Lipton et al. [2003] and Lipton and Markakis [2004]. For fixed r, the first is quasipolynomial in the number n of strategies per player (to be specific, proportional to $n^{\log n}$) but exponential in $2^{|A|}$. The second is polynomial in $|A|$ but exponential in n. Both of these bounds are far from polynomial. However, until two years ago the question of whether any r-NASH problem was PPAD-complete remained open.

Now, after a rapid flurry of results, we know the answer for all r. Goldberg and Papadimitriou [2006] showed that if 4-PLAYER NASH were in FP, then so would be the problems of finding Nash equilibria for any game with a bounded number of players and for any graphical game with bounded degree. Unfortunately, as shown at about the same time by Daskalakis et al. [2006], 4-PLAYER NASH is PPAD-complete. The proof used a reduction from a new 3-dimensional discrete problem, based on Brouwer's fixed point theorem, to a particular type of graphical game, with the Brouwer problem proved PPAD-complete by a transformation from END OF THE LINE. Shortly after these papers were written (but before they were published), both Chen and Deng [2005] and Daskalakis and Papadimitriou [2005] showed that 3-PLAYER NASH is PPAD-complete, the first by a direct reduction from 4-PLAYER NASH and the second by using a variant on the proof of Daskalakis et al.

[2006]. Finally, in a paper presented at last year's FOCS conference, Chen and Deng [2006b] showed that the original open problem, 2-PLAYER NASH, was PPAD-complete via a yet-more-complicated reduction from the same Brouwer problem. Moreover, this result could be strengthened to hold even with an inverse polynomial (rather than inverse exponential) accuracy parameter, thus ruling out the possibility of an FPTAS for 2-PLAYER NASH unless PPAD = FP [Chen et al. 2006]. (Technically, all these completeness results were proved in the context of a slightly more restrictive notion of "approximate Nash equilibrium" than the one that I described here and that was used in the algorithmic results that I discussed. The two notions, however, are equivalent with respect to PPAD-completeness [Chen et al. 2006; Daskalakis et al. 2007].)

I conclude this column by briefly describing three other subclasses of TFNP that were introduced by Papadimitriou [1994]. Each exploits a distinct combinatorial existence theorem to insure that its members are in TFNP. Each contains PPAD as a subclass and, like PPAD, consists of all problems in TFNP that have TFNP reductions to a member of a specified base class.

PPADS. Here the base class consists of problems with the same sorts of instances as those in PPAD, but with the goal of finding a sink (solution with indegree 1 and outdegree 0), not finding either a sink or a source other than y_0. Like PPAD, PPADS is contained in TFNP because a directed graph with maximum in- and outdegree 1 that has a source must have a sink. The two classes were claimed to be equal in Papadimitriou's proceedings paper [1990], but this was retracted in the journal version [1994]. The variant of 3D SPERNER in which we look for a unit cube containing a panchromatic tetrahedron whose prime facet, when viewed from inside, has the positive orientation, is complete for PPADS [Beame et al. 1998]. In addition, Friedl et al. [2006] presents a Sperner problem on a locally 2-dimensional surface that is complete for this class.

PPA. This is the undirected analogue of PPAD. Problems in the base class for PPA, instead of having successor and predecessor functions, have a single *neighbor* function f such that $f(x, y)$ is a set consisting of *two* solutions, and our implicit graph has an edge between y and y' if $y \in f(x, y')$ and $y' \in f(x, y)$. The initial solution is required to be a leaf (degree-1 vertex) in this graph, and our goal is to find a second solution that is a leaf. Since no vertex can have degree exceeding 2 in this graph, such a solution must exist by the same parity argument we mentioned in our discussion of Sperner's Lemma. Grigni [2001] presents a Sperner problem on a non-orientable 3-dimensional manifold that is complete for this class.

PPP. Problems in the base class for PPP involve a single function f that takes solutions to solutions. Our goal is to find either a solution y that maps to the initial solution (i.e., $f(x, y) = y_0$) or two solutions y and y' such that $f(x, y) = f(x, y')$. Such a solution exists by the pigeonhole principle (PPP abbreviates "Polynomial Pigeonhole Principle").

Certain containment relationships between these classes and PPAD are easy to prove. In particular, we have PPAD $\subseteq$ PPADS $\subseteq$ PPP and PPAD $\subseteq$ PPA. Certain other containments may be less likely, or at least harder to prove, since there are relativized worlds in which they do not hold. Indeed, for a "generic" oracle A we

have $\mathrm{PPA}^A \not\subseteq \mathrm{PPP}^A$, $\mathrm{PPP}^A \not\subseteq \mathrm{PPADS}^A$, and $\mathrm{PPADS}^A \not\subseteq \mathrm{PPA}^A$ (and hence $\mathrm{PPP}^A \not\subseteq \mathrm{PPA}^A$) [Beame et al. 1998].

Generic oracles, introduced by Blum and Impagliazzo [1987], are unrelated to the concept of the generic complete problem discussed in Section 2. They exploit forcing arguments of the type used by Cohen [1964] in his proof of the independence of the continuum hypothesis (see also Fenner et al. [2003]). Results that hold for one generic oracle hold for all of them, so they give consistent advice about the relations between classes, as do random oracles (see Column 4 [1982]). Moreover, it can be argued [Blum and Impagliazzo 1987] that generic oracles do not suffer from some of the drawbacks of random oracles, which among other things provide a source of randomness for free. Unfortunately, as with random oracles, the advice of generic oracles is not always correct. For example, although IP = PSPACE [Lund et al. 1992; Shamir 1992], the two classes differ relative to a generic oracle [Foster 1993], just as they do for a random oracle [Fortnow and Sipser 1988]. Thus perhaps the best we can glean from the oracle results of Beame et al. [1998] is that standard techniques will not suffice to prove the denied containments, a conclusion we could already gather from an ordinary oracle result.

As a final question, consider the relation between the classes of this section and the class PLS discussed in Section 3. We should first note that $\mathrm{PLS} \cap \mathrm{PPAD} - \mathrm{FP}$ does not at present appear to be empty. In particular, SIMPLE STOCHASTIC GAME and the other game problems from Section 2 can be shown to be both in PLS [Condon 1993; Yannakakis 1990] and in PPAD [Juba 2005], although these problems do not appear to be complete for either class. Moreover, there is a superficial similarity between PLS and PPADS, given that in both we are searching for a sink in a directed graph. However, there seem to be serious obstacles to simulating one class by the other. In PLS, vertices may have indegree exceeding 1, but sinks are guaranteed by the fact that the underlying graph is acyclic. In PPADS the graphs are not necessarily acyclic, but sinks are guaranteed because there is a source and no vertex has indegree exceeding 1.

Thus it is perhaps not surprising that relative to a generic oracle A, PLS does not contain PPAD, which implies that it does not contain PPADS, PPA, or PPP, either [Morioka 2001]. Moreover, relative to a generic oracle, PLS is not contained in PPA, although its containment in PPP relative to such an oracle has not yet been ruled out [Buresh-Oppenheim and Morioka 2004]. Note that these results in addition imply (for what it is worth) that relative to a generic oracle, TFNP is not contained in FP, which has been the working hypothesis of this column.

ACKNOWLEDGMENT.

In preparing this column I received valuable advice, pointers, and technical information from David Applegate, Aaron Archer, Paul Beame, Jon Bentley, Anne Condon, Costis Daskalakis, Paul Goldberg, Brendan Juba, Howard Karloff, Vangelis Markakis, Christos Papadimitriou, Suresh Venkatasubramanian, Mihalis Yannakakis, and Uri Zwick.

REFERENCES

Agrawal, M., Kayal, N., and Saxena, N. 2004. PRIMES is in P. *Ann. Math. 160*, 781–793.

Álvarez, C., Gabarró, J., and Serna, M. 2005. Pure Nash equilibria in games with a large number of actions. In *Proceedings of the 30th International Symposium on Mathematical*

Foundations of Computer Science. Lecture Notes in Computer Science, vol. 3618. Springer-Verlag, Berlin, 95–106.

Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., and Pandit, V. 2004. Local search heuristics for k-median and facility location problems. *SIAM J. Comput. 33*, 544–562. (Preliminary version in *Proceedings of the 33rd Annual ACM Symposium of Computer Science*, ACM, New York, 2001, 21–29.)).

Beame, P., Cook, S., Edmonds, J., Impagliazzo, R., and Pitassi, T. 1998. The relative complexity of NP search problems. *J. Comput. Syst. Sci. 57*, 3–19.

Blum, M. and Impagliazzo, R. 1987. Generic oracles and oracle classes. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Los Alamitos, Calif., 118–126.

Brouwer, L. E. J. 1910. Über eneindeutige, stetige Transformationen von Flächen in sich. *Math. Ann. 69*, 176–180.

Buresh-Oppenheim, J. and Morioka, T. 2004. Relativized NP search problems and propositional proof systems. In *Proceedings of the 19th Annual IEEE Conference on Computational Complexity*. IEEE Computer Society, Los Alamitos, Calif., 54–67.

Chen, X. and Deng, X. 2005. 3-Nash is PPAD-complete. Tech. Rep. TR05-134, Electronic Colloquium on Computational Complexity.

Chen, X. and Deng, X. 2006a. On the complexity of 2D discrete fixed point problems. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming*. Lecture Notes in Computer Science, vol. 4051. Springer-Verlag, Berlin, 489–500.

Chen, X. and Deng, X. 2006b. Settling the complexity of two-player Nash equilibrium. In *Proceedings of the 47th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Los Alamitos, Calif., 261–270.

Chen, X., Deng, X., and Teng, S.-H. 2006. Computing Nash equilibria: Approximation and smoothed complexity. In *Proceedings of the 47th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Los Alamitos, Calif., 603–612.

Cohen, P. J. 1964. The independence of the continuum hypothesis. *Proc. Nat. Acad. Sci 51*, 105–110.

Condon, A. 1992. The complexity of stochastic games. *Inform. Comp. 96*, 203–224.

Condon, A. 1993. On algorithms for simple stochastic games. In *Advances in Computational Complexity Theory*, J.-Y. Cai, Ed. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 13. American Mathematical Society, Providence, RI, 51–73.

Daskalakis, C., Goldberg, P. W., and Papadimitriou, C. H. 2006. The complexity of computing a Nash equilibrium. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*. ACM, New York, 71–78.

Daskalakis, C., Goldberg, P. W., and Papadimitriou, C. H. 2007. The complexity of computing a Nash equilibrium. Submitted journal version of Daskalakis et al. [2006], Goldberg and Papadimitriou [2006], and Daskalakis and Papadimitriou [2005].

Daskalakis, C. and Papadimitriou, C. H. 2005. Three-player games are hard. Tech. Rep. TR05-139, Electronic Colloquium on Computational Complexity.

Ehrenfeucht, A. and Mycielski, J. 1979. Positional strategies for mean payoff games. *Int. J. Game Theory 8*, 109–113.

Emerson, E. A. and Jutla, C. S. 1991. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Los Alamitos, Calif., 368–377.

Englert, M., Röglin, H., and Vöcking, B. 2007. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, Philadelphia, 1295–1304.

Fabrikant, A., Papadimitriou, C. H., and Talwar, K. 2004. The complexity of pure Nash equilibria. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*. ACM, New York, 604–612.

Fenner, S., Fortnow, L., Kurtz, S., and Li, L. 2003. An oracle builder's toolkit. *Inform. Comp. 182*, 95–136.

Fortnow, L. and Sipser, M. 1988. Are there interactive protocols for co-NP languages? *Inform. Proc. Lett. 28*, 249–251.

Foster, J. A. 1993. The generic oracle hypothesis is false. *Inform. Proc. Lett. 45*, 59–62.

Friedl, K., Ivanyos, G., Santha, M., and Verhoeven, Y. F. 2006. Locally 2-dimensional Sperner problems complete for polynomial parity argument classes. In *Proceedings of the 6th Italian Conference on Algorithms and Complexity*. Lecture Notes in Computer Science, vol. 3998. Springer-Verlag, Berlin, 380–391.

Goldberg, P. W. and Papadimitriou, C. H. 2006. Reducibility among equilibrium problems. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*. ACM, New York, 61–70.

Gottlob, G., Greco, G., and Scarcello, F. 2003. Pure Nash equilibria: Hard and easy games. In *Proceedings of the 9th Conference on Theoretical Aspects of Rationality and Knowledge*. ACM, New York, 215–230.

Grigni, M. 2001. A Sperner lemma complete for PPA. *Inform. Proc. Lett. 77*, 255–259.

Gurvich, V. A., Karzanov, A. V., and Khachiyan, L. G. 1988. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *U.S.S.R. Comp. Math. & Math. Physics 28*, 85–91.

Homan, C. M. and Thakur, M. 2003. One-way permutations and self-witnessing languages. *J. Comput. Syst. Sci. 67*, 608–622.

Hopfield, J. J. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci. 79*, 2554–2558.

Johnson, D. S. and McGeoch, L. A. 1997. The traveling salesman problem: A case study in local optimization. In *Local Search in Combinatorial Optimization*, E. Aarts and J. K. Lenstra, Eds. John Wiley & Sons, Chichester, 215–310. Preliminary version available at `http://www.research.att.com/~dsj/papers`.

Johnson, D. S., Papadimitriou, C. H., and Yannakakis, M. 1988. How easy is local search? *J. Comput. Syst. Sci. 37*, 79–100.

Juba, B. 2005. On the hardness of simple stochastic games. M.S. Thesis, Carnegie-Mellon University. Currently available from `http://people.csail.mit.edu/bjuba/`.

Jurdziński, M. 1998. Deciding the winner in parity games is in UP ∩ co-UP. *Inform. Proc. Lett. 68*, 119–124.

Jurdziński, M., Paterson, M., and Zwick, U. 2006. A deterministic subexponential algorithm for solving parity games. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, Philadelphia, 117–123.

Kakutani, S. 1941. A generalization of Brouwer's fixed point theorem. *Duke Math. J. 8*, 457–459.

Karzanov, A. V. and Lebedev, V. N. 1993. Cyclical games with prohibitions. *Math. Prog. 60*, 277–293.

Kernighan, B. and Lin, S. 1970. An efficient heuristic for partitioning graphs. *Bell Syst. Tech. J. 49*, 291–307.

Khachiyan, L. G. 1979. A polynomial algorithm in linear programming. *Dokl. Akad. Nauk. SSSR 20*, 191–194.

Klee, V. and Minty, G. J. 1972. How good is the simplex algorithm? In *Inequalities III*, O. Shisha, Ed. Academic Press, New York, 159–175.

Korupolu, M. R., Plaxton, C. G., and Rajaraman, R. 2000. Analysis of a local search heuristic for facility location problems. *J. Algorithms 37*, 146–188.

Krentel, M. W. 1989. Structure in locally optimal solutions. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Los Alamitos, Calif., 216–221.

Lemke, C. E. 1965. Bimatrix equivalence points and mathematical programming. *Mgmt. Sci. 11*, 681–689.

Lenstra, A. K. and Lenstra, H. W., Eds. 1993. *Development of the Number Field Sieve*. Lecture Notes in Mathematics, vol. 1554. Springer-Verlag, Berlin.

Lin, S. and Kernighan, B. 1973. An effective heuristic algorithm for the traveling salesman problem. *Operations Research 21*, 972–989.

Lipton, R. J. and Markakis, E. 2004. Nash equilibria via polynomial equations. In *Latin 2004: Theoretical Informatics*. Lecture Notes in Computer Science, vol. 2976. Springer-Verlag, Berlin, 413–422.

Lipton, R. J., Markakis, E., and Mehta, A. 2003. Playing large games using simple strategies. In *Proceedings of the 4th ACM Conference on Electronic Commerce*. ACM, New York, 36–41.

Lueker, G. 1976. Manuscript, Princeton University.

Lund, C., Fortnow, L., Karloff, H., and Nisan, N. 1992. Algebraic methods for proof systems. *J. ACM 39*, 859–868.

Megiddo, N. and Papadimitriou, C. H. 1991. On total functions, existence theorems and computational complexity. *Theor. Comput. Sci. 81*, 317–324.

Morioka, T. 2001. The classification of search problems and their definability in bounded arithmetic. M.S. Thesis, University of Toronto. Also available as Tech. Rep. TR01-82, Electronic Colloquium on Computational Complexity.

Nash, J. F. 1951. Non-cooperative games. *Ann. Math. 54*, 286–295.

Papadimitriou, C. H. 1990. On graph-theoretic lemmata and complexity classes. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computing*. IEEE Computer Society, Los Alamitos, Calif., 794–801. (Preliminary version of Papadimitriou [1994]).

Papadimitriou, C. H. 1992. The complexity of the Lin-Kernighan heuristic for the traveling salesman problem. *SIAM J. Comput. 21*, 450–465.

Papadimitriou, C. H. 1994. The complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci. 48*, 498–532. (Journal version of Papadimitriou [1990].).

Papadimitriou, C. H., Schäffer, A. A., and Yannakakis, M. 1990. On the complexity of local search. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*. ACM, New York, 438–445.

Papadimitriou, C. H. and Steiglitz, K. 1978. Some examples of difficult traveling salesman problems. *Oper. Res. 26*, 434–443.

Pomerance, C. 1996. A tale of two sieves. *AMS Notices 43*, 1473–1485.

Pratt, V. 1975. Every prime has a succinct certificate. *SIAM J. Comput. 4*, 214–220.

Puri, A. 1995. Theory of hybrid systems and discrete event systems. Ph.D. thesis, University of California, Berkeley.

Rosenthal, R. W. 1973. A class of games possessing pure-strategy Nash equilibria. *Int. J. Game Theory 2*, 65–67.

Schäffer, A. A. and Yannakakis, M. 1991. Simple local search problems that are hard to solve. *SIAM J. Comput. 20*, 56–87.

Shamir, A. 1992. IP = PSPACE. *J. ACM 39*, 869–877.

Sipser, M. 1982. On relativization and the existence of complete sets. In *Proceedings of the 9th International Colloquium on Automata, Languages and Programming*. Lecture Notes in Computer Science, vol. 140. Springer-Verlag, Berlin, 523–531.

Sperner, E. 1928. Neuer beweis für die Invarianz der Dimensionzahl und des Gebietes. *Abh. math. Sem. Univ. Hamburg 6*, 265–272.

Vöcking, B. 2006. Congestion games: Optimization in competition. In *Proceedings of the 2nd Algorithms and Complexity in Durham Workshop*, H. Broersma, S. Dantchev, M. Johnson, and S. Szeider, Eds. Kings College Publications, London, 9–20.

Yannakakis, M. 1990. The analysis of local search problems and their heuristics. In *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computing*. Lecture Notes in Computer Science, vol. 415. Springer-Verlag, Berlin, Germany, 298–311.

Yannakakis, M. 1997. Computational complexity. In *Local Search in Combinatorial Optimization*, E. Aarts and J. K. Lenstra, Eds. John Wiley & Sons, Chichester, 19–55.

Zwick, U. and Paterson, M. 1996. The complexity of mean payoff games on graphs. *Theor. Comput. Sci. 158*, 343–359.