available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cosrev

Survey

Equilibria, fixed points, and complexity classes☆

Mihalis Yannakakis

Department of Computer Science, Columbia Unversity, United States

ARTICLE INFO

Article history:
Received 24 March 2009
Received in revised form
27 March 2009
Accepted 27 March 2009

ABSTRACT

Many models from a variety of areas involve the computation of an equilibrium or fixed point of some kind. Examples include Nash equilibria in games; market equilibria; computing optimal strategies and the values of competitive games (stochastic and other games); stable configurations of neural networks; analysing basic stochastic models for evolution like branching processes and for language like stochastic context-free grammars; and models that incorporate the basic primitives of probability and recursion like recursive Markov chains. It is not known whether these problems can be solved in polynomial time. There are certain common computational principles underlying different types of equilibria, which are captured by the complexity classes PLS, PPAD, and FIXP. Representative complete problems for these classes are, respectively, pure Nash equilibria in games where they are guaranteed to exist, (mixed) Nash equilibria in two-player normal form games, and (mixed) Nash equilibria in normal form games with three (or more) players. This paper reviews the underlying computational principles and the corresponding classes.

1. Introduction

Many situations involve the computation of an equilibrium or a stable configuration of some sort in a dynamic environment. Sometimes it is the result of individual agents acting on their own noncompetitively but selfishly (e.g., Nash and other economic equilibria), sometimes it is agents acting competitively against each other (and perhaps nature/chance), and sometimes the equilibrium is the limit of an iterative process that evolves in some direction until it settles. Often the sought objects can be described mathematically as the fixed points of an equation $x = F(x)$.

Many models and problems from a broad variety of areas are of this nature. Examples include: Nash equilibria in games; market equilibria; computation of optimal strategies and the values of competitive games (stochastic and other games); stable configurations of neural networks; analysis of basic stochastic models for evolution like branching processes, and for language like stochastic context-free grammars; and models that incorporate the basic primitives of probability and recursion like recursive Markov chains. Most of these models and problems have been studied mathematically for a long time, leading to the development of rich theories. Yet, some of their most basic algorithmic questions are still not resolved; in particular, it is not known whether they can be solved in polynomial time.

Despite the broad diversity of these problems, there are certain common computational principles that underlie many of these different types of problems, which are captured by the complexity classes PLS, PPAD, and FIXP. In this paper we will review these principles, the corresponding classes, and the types of problems they contain.

☆ Work supported by NSF Grant CCF-0728736. A preliminary version of this paper appeared in Proc. 25th STACS.
E-mail address: mihalis@cs.columbia.edu.

1574-0137/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.cosrev.2009.03.004

is (a subset of) $\mathbb{R}^d$, i.e. the solutions are real-valued vectors of finite dimension d, where d is polynomial in the input size. For some problems, even though the underlying space is continuous, there are always rational-valued solutions. Linear Programming (LP) is such an example (technically, to consider LP as a total search problem, we should include 'infeasible' and 'unbounded' as possible solutions to cover these cases); a feasible, bounded LP (with rational data) always has a rational-valued optimal solution, and indeed one that has size bounded by a polynomial in the input size. In these cases where there are always rational solutions, we can redefine the problem to require such a solution, i.e., restrict $\mathrm{Sol}(I)$ to its rational-valued subset.

In many problems, however, the solutions are inherently irrational, so we cannot output them explicitly; for example, Nash equilibria for games with three or more players, probabilities of branching processes, values of stochastic games, and so forth. One approach would be to study the complexity of the problems in a real computation model, such as the model of [1]. However, we will stay with the standard Turing machine model. In this model we can compute only some desired finite information about a solution, for example, approximate it to a desired precision, or compute some predicate about it, for example compare a coordinate to a given threshold. That is, in this approach, the original search problem is mapped to another discrete search problem, whose complexity can be studied in the standard Turing model. Different types of information result in different discrete problems. Some basic types of such discrete versions are the following:

1. *Decision problem:* Given instance I, rational r, and coordinate i, output the truth value of the predicate $x_i \geq r$ (or $x_i \leq r$, etc.) for a solution $x \in \mathrm{Sol}(I)$ (any solution).
2. *Partial computation:* Given instance I and integer $k > 0$ in unary, output the k most significant bits of a solution $x \in \mathrm{Sol}(I)$.
3. *Approximation problem:* Given instance I and rational $\epsilon > 0$ (in binary), output a vector x that is within (additive) ϵ of a solution, i.e. there is a $x^* \in \mathrm{Sol}(I)$ such that $|x_i - x_i^*| \leq \epsilon$ for all i.

The decision version corresponds to the standard way of turning problems with output (for example optimization problems) to languages (yes/no problems) to study their complexity. For example, if the original problem is to compute the probability p of an event in a stochastic model (for instance, the extinction probability of a branching process), then the decision problem includes questions like 'is $p = 1$?', 'is $p \geq 1/2$?', while the approximation problem concerns the approximation of p to any desired precision. Note that for search problems with multiple solutions, any one of the solutions can be used to answer easily the associated discrete problems listed above. For example, if an instance I has a solution with $x_i \geq r$ and another solution with $x_i < r$, then both true and false are acceptable answers for the predicate $x_i \geq r$ in the decision problem (they are both truth values of some solution), as opposed to the *Existence problem*, "Does there exist a solution that satisfies $x_i \geq r$?", where only true would be the correct answer. For problems with unique solutions, the Decision and the Existence problems coincide, but for search problems with multiple solutions, they differ and the Existence problem can be in general harder. For example, in the Nash equilibrium problem for two-player games, the Existence problem is NP-hard, whereas the Decision problem (and the search problem itself) cannot be NP-hard unless NP = coNP.

Ideally we would like to solve these (discrete) problems in polynomial time in their inputs, for example to approximate a solution within ϵ in polynomial time in $|I|$ and $\log(1/\epsilon)$. Of course, if this is not possible, we would like to get as good an approximation as possible in polynomial time in $|I|$.

Note that these (discrete) versions of a search problem may have different complexities. For example, consider Semidefinite Programming; the approximation problem can be solved in polynomial time, but we do not know how to solve the decision problem (and very likely it is not in P). For problems that have unique solutions and polynomially bounded domains, if we can solve the decision problem in polynomial time then we can obviously also solve the others using binary search; the converse may not be true, however. Note also that for problems that have rational solutions of polynomially size, if we can solve the approximation problem in polynomial time (in $|I|$ and $\log(1/\epsilon)$) then we can also compute an exact solution.

Reductions between search problems

In the case of discrete search problems, a reduction from problem A to problem B consists of two polynomial-time computable functions: a function f that maps every instance I of A to an instance $f(I)$ of B, and a second function g that maps each solution $y \in \mathrm{Sol}(f(I))$ of the instance $f(I)$ of B to a solution $x \in \mathrm{Sol}(I)$ of the instance I of A. For search problems with real-valued solutions we have to specify what kind of functions g are allowed, since our model is the standard Turing machine model which does not operate directly on real numbers. The main criterion is that the definition should enable us to transfer easily back from B to A solutions for the discrete problems of interest associated to search problems, e.g., the decision and approximation problems. It is sufficient in all the cases discussed in this paper to restrict the reverse function g to have a particularly simple form: a *separable linear* transformation with polynomial-time computable rational coefficients; that is, $x = g(y)$, where each $g_i(y)$ is of the form $a_i y_j + b_i$ for some j, where a_i, b_i are rationals computable from I in polynomial time. Call such a reduction, consisting of a polynomial-time computable f and a separable linear g, an *SL-reduction*. It is easy to see that, if A SL-reduces to B, then the decision and approximation problems for A reduce (now in the standard discrete sense) to the corresponding problems for B. Furthermore, if the coefficients a_i, b_i of g are powers of 2 (which is the case in all reductions here), then the partial computation problem for A reduces to that of B.

We will define as we along in the rest of the paper the models, concepts and problems in various areas, and point to some of the relevant references. For overviews of various parts of game theory and their algorithmic aspects see [66,72]. A nice exposition of the complexity of total search problems (including problems in PLS and PPAD) is given in [69]. These references predate and do not cover FIXP and the recent results on the computation and approximation of actual equilibria, and more generally fixed points in different areas.

3. Discrete, pure equilibria and the class PLS

Consider the following *neural network* model [2]: We have an undirected graph $G = (V, E)$ with a positive or negative weight $w(e)$ on each edge $e \in E$ (we can consider missing edges as having weight 0) and a threshold $t(v)$ for each node $v \in V$. A configuration of the network is an assignment of a state $s(v) = +1$ ('on') or -1 ('off') to each node $v \in V$. A node v is *stable* (or 'happy') if $s(v) = 1$ and $\sum_u w(v,u)s(u) + t(v) \geq 0$, or $s(v) = -1$ and $\sum_u w(v,u)s(u) + t(v) \leq 0$, i.e. the state of v agrees with the sign of the weighted sum of its neighbors plus the threshold. A configuration is *stable* if all the nodes are stable. A priori it is not obvious that such a configuration exists; in fact for directed networks there may not exist any stable configuration. However, every undirected network has at least one (or more) stable configuration [2]. Furthermore, such a configuration can be obtained by a simple dynamic process, which starts with an arbitrary configuration, and then, in each step, switches the state of one unstable node (any one) until the network stabilizes; this process is guaranteed to eventually converge in a finite number of steps to a stable configuration, no matter which unstable node is switched in each step. (It is important that updates be asynchronous, one node at a time; simultaneous updates can lead to oscillations.)

To show the existence of a stable configuration and convergence of the process, Hopfield introduced a value function (or 'potential' or 'energy') on configurations, $p(s) = \sum_{(v,u)\in E} w(v,u)s(v)s(u) + \sum_{v\in V} t(v)s(v)$. If v is an unstable node in configuration s, then switching its state results in a configuration s' with strictly higher value, $p(s') = p(s) + 2|\sum_u w(v,u)s(u) + t(v)| > p(s)$. One immediate consequence of this fact is that every network has (one or more) stable configurations; for example, the configurations with maximum value are stable. A second consequence is that every execution of the dynamic process must converge in a finite number of steps to a stable configuration. The reason is that there is a finite number of configurations (namely, $(2^{|V|})$), and the same configuration cannot repeat in an execution because the value is monotonically strictly increasing.

The *stable configuration problem* is the following: Given a neural network, compute a stable configuration. This is a total search problem, as there are always (one or more) stable configurations, and any one of them is an acceptable output. Note that the value (energy) function $p(s)$ is not intrinsic to the problem statement; it is simply a *tool* for showing the existence of a solution (and convergence of the dynamic process). The configurations with maximum value are certainly solutions, but they are not the only ones. In fact, computing a configuration with maximum value is an NP-hard problem; it includes the Max-Cut problem as a special case, namely, the case where all thresholds are 0 and all edge weights are -1. However, this does not necessarily mean that one cannot compute in polynomial time *some* stable configuration; indeed, if the weights are -1, or more generally, polynomially bounded in magnitude, then the dynamic process always converges in polynomial time, since the value increases in each step.

Although the stable configuration problem does not call a priori for any optimization, the problem can be viewed equivalently as one of *local* optimization: compute a configuration s whose value $p(s)$ cannot be increased by switching the state of any single node. Local search is a common, general approach for tackling hard optimization problems. In a combinatorial optimization problem, every instance I has an associated finite set $S(I)$ of (candidate) solutions, and every solution $s \in S(I)$ has a rational value or cost $p_I(s)$ that is to be maximized or minimized. In local search, each solution $s \in S(I)$ has in addition an associated neighborhood $N_I(s) \subseteq S(I)$; a solution is locally optimal if it does not have any (strictly) better neighbor, i.e. one with higher value or lower cost. The search problem is to compute a locally optimal solution, i.e. the 'acceptable' set $\mathrm{Sol}(I)$ for an instance I of the local search problem consists of all locally optimal solutions. One way to compute such a solution is to use a standard local search algorithm: start from an initial solution, and keep moving to a better neighbor as long as there is one, until the algorithm reaches a local optimum. The complexity class PLS (Polynomial Local Search) was introduced in [3] to capture the inherent complexity of local optima for usual combinatorial problems, where each step of the local search algorithm can be done in polynomial time. Even though each step takes polynomial time, the number of steps can be potentially exponential, and in fact for many problems we do not know how to compute even locally optimal solutions in polynomial time.

Formally, a problem Π is in PLS if solutions are (represented by strings that are) polynomially bounded in the input size, and there are polynomial-time algorithms for the following tasks: (a) given string I, test whether I is an instance of Π and if so compute a (initial) solution in $S(I)$, (b) given I, s, test whether $s \in S(I)$, and if so compute its value $p_I(s)$, (c) given I, s, test whether s is a local optimum, and if not, compute a better neighbor $s' \in N_I(s)$. A notion of PLS reduction and completeness were introduced to relate the problems. A PLS reduction from problem A to problem B is simply a polynomial-time reduction between discrete search problems (i.e. with finite solution spaces), as defined in Section 2. A problem $\Pi \in PLS$ is PLS-complete if every problem in PLS reduces to Π. A number of well-studied combinatorial optimization problems (e.g. Graph Partitioning, TSP, Max Cut, Max Sat, etc.) with common neighborhood structures (both sophisticated ones like Kernighan-Lin and simple ones like flipping a bit) have been shown to be PLS-complete, and thus locally optimal solutions can be computed efficiently for anyone of them iff they can be computed for all PLS problems. For a detailed survey and bibliography see [4]. In particular, the stable configuration problem is PLS-complete (and is complete even if all thresholds are 0 and all weights are negative, i.e. all connections are repulsive; this special case corresponds to Weighted Max Cut) [5].

It is worth stressing several points:

1. The search problem asks to compute any local optimum, not a specific one like the best, which is often NP-hard.
2. Given an instance I, we can always guess a solution s, and verify in polynomial time that it is indeed a solution ($s \in S(I)$) and it is locally optimal. Hence PLS is somewhere between P and TFNP (total search problems in NP). Such problems cannot be NP-hard (under Cook reductions) unless NP = coNP. This means in particular that (assuming NP $\neq$ coNP), we

cannot use NP-hardness to justify our inability so far to find polynomial-time algorithms for these problems.

3. We are interested in the inherent complexity of the search problem itself *by any algorithm whatsoever*, not necessarily the standard local search algorithm, which often has exponential running time. For example, Linear Programming can be viewed as a local search problem (where local optima = global optima) with Simplex as the local search algorithm. We know that Simplex under many pivoting rules is exponential, yet the problem itself can be solved in polynomial time by completely different methods (Ellipsoid, Karmakar). In fact, many common local search problems are complete under a restricted type of reduction called *tight* PLS-reduction, which allows us to conclude that the corresponding standard local search algorithm is exponential. For example, in the neural network model, the dynamic process where unstable nodes switch their state iteratively until the network stabilizes takes for some networks and for some (in fact for most) initial configurations exponential time to converge, no matter which unstable node is switched in each step. Furthermore, the computational problem: given a network and initial configuration, compute a stable configuration (any one) that can result from this process, is a PSPACE-complete problem.

Another type of equilibrium problems that can be placed in PLS concerns finding pure Nash equilibria for games where they are guaranteed to exist. A (finite) *game* has a finite set k of players, and each player, $i = 1, \ldots, k$, has a finite set S_i of *pure strategies* and a *payoff* (utility) function U_i on the product strategy space $S = \Pi_i S_i$; we assume for computational purposes that U_i takes rational values. A pure *strategy profile* s is a member of S, i.e. a choice of a pure strategy $s_i \in S_i$ for each player. It is a *pure Nash equilibrium* if no player can improve his payoff by switching unilaterally to another pure strategy; that is, if (s_{-i}, s'_i) denotes the profile where player i plays strategy $s'_i \in S_i$ and the other players play the same strategy as in s, then $U_i(s) \geq U_i(s_{-i}, s'_i)$ for every i and every $s'_i \in S_i$. Not every game has a pure Nash equilibrium. A *mixed strategy* for player i is a probability distribution on S_i. Letting M_i denote the set of mixed strategies for player i, the set of mixed strategy profiles is their product $M = \Pi_i M_i$; i.e., a mixed strategy profile is a non-negative vector x of length $\sum_i |S_i|$ (i.e. its entries are indexed by all the players' pure strategies) that is a probability distribution on the set of pure strategies of each player. The (expected) payoff $U_i(x)$ of x for player i is $\sum x_{1,j_1} \ldots x_{k,j_k} U_i(j_1, \ldots, j_k)$, where the sum is over all tuples $(j_1, \ldots, j_k)$ such that $j_1 \in S_1, \ldots, j_k \in S_k$, and $x_{i,j}$ is the entry of x defining the probability with which player i plays strategy j. A (mixed) *Nash equilibrium* (NE) is a (mixed) strategy profile x^* such that no player can increase its payoff by switching to another strategy unilaterally. Nash's fundamental theorem asserts that every finite game has at least one Nash equilibrium [6].

For example, a neural network can be viewed as a game with one player for each node, each player has two pure strategies, $+1$, -1 (corresponding to the two states), and its payoff function has two values, 1 (happy) and 0 (unhappy), depending on its state and that of its adjacent nodes. The stable configurations of the network are exactly the pure Nash equilibria of the game. This game is a case of a *graphical game*: players correspond to nodes of a graph and the payoff function of a player depends only on its own strategy and that of its neighbors. General graphical games may not have pure Nash equilibria. For an overview of graphical games see [7].

There is a class of games, *congestion games*, in which there is always a pure equilibrium. In a congestion game, there are k players, a finite set R of resources, the pure strategy set $S_i \subseteq 2^R$ of each player is a family of subsets of the resources, and each resource $r \in R$ has an associated cost function $d_r : \{0, \ldots, k\} \to Z$. If $s = (s_1, \ldots, s_k)$ is a pure strategy profile, the congestion $n_r(s)$ of a resource r is the number of players whose strategy contains r; the cost (negative payoff) of a player i is $\sum_{r \in s_i} d_r(n_r(s))$. Rosenthal showed that every congestion game has a pure equilibrium [8]. In fact, the iterative process where, in each step, if the current pure strategy profile is not at equilibrium, a player with a suboptimal strategy switches to a strategy with a lower cost (while other players keep the same strategy) does not repeat any profile and thus converges in a finite number of steps to an equilibrium. The proof is by introducing a potential function $p(s) = \sum_{r \in R} \sum_{i=1}^{n_r(s)} d_r(i)$ and showing that if a (pure) strategy profile s is not an equilibrium then switching the strategy of any player to a lower cost strategy results in a reduction of the potential function by the same amount. Thus, the pure equilibria are exactly the local optima of the potential function $p(s)$ with respect to the neighborhood that switches the strategy of a single player.

Computing a pure equilibrium of a congestion game can be viewed as a local search problem, and it is in PLS provided that the cost functions d_r of the resources are polynomial-time computable, and the strategy sets S_i are given explicitly or at least one can determine efficiently whether a player can improve his strategy for a given profile. Furthermore, Fabrikant et al. [9] showed that the pure equilibrium problem for congestion games is PLS-complete. They showed that it is complete even in the case of *network congestion games*, where the resources are the edges of a given directed graph, each player i has an associated source s_i and target node t_i and its set S_i of pure strategies is the set of $s_i - t_i$ paths; the cost function d_r of each edge r represents the delay as a function of the number of paths that use the edge. PLS-completeness holds even for linear delay functions on the edges [10] and for computing approximate pure equilibria [11]. As with other PLS-complete problems, a consequence of the reductions, which are tight, is that the iterative local improvement algorithm, where the players switch to better strategies, can take exponential time to converge. For an overview of results on congestion games see [12].

There are several other games which are in PLS and not known to be in P, and which are not known (and not believed to be) PLS-complete. These are not one-shot games, but rather they are dynamic games played iteratively over time (like chess, backgammon, etc.). There are two main types of payoff for the players in such games: in one type, the payoff of a history is an aggregation of rewards obtained in the individual steps of the history combined via some aggregation function, such as average reward per step or a discounted sum of the rewards; in the other type, the payoff obtained depends on the properties of the history. We will discuss three such dynamic games in this section, and some more in the following sections.

A *simple stochastic game* [13] is a two-player game played on a directed graph $G = (V, E)$ whose nodes represent the positions of the game, and the edges represent the possible moves. The sinks are labeled 1 or 2 and the nonsink nodes are partitioned into three sets, V_r (random nodes), V_1 (max or player 1 nodes), V_2 (min or player 2 nodes); the edges (u, v) out of each random node u are labeled with probabilities p_{uv} (assumed to be rational for computational purposes) that sum to 1. Play starts at some initial node (position) and then moves in each step along the edges of the graph; at a random node the edge is chosen randomly, at a node of V_1 it is chosen by player 1, and at a node of V_2 it is chosen by player 2. If the play reaches a sink labeled 1, then player 1 is the winner, while if it reaches a sink labeled 2 or it goes on forever, then player 2 is the winner. The goal of player 1 is to maximize her probability of winning, and the goal of player 2 is to minimize it (i.e. maximize his own winning probability).

Simple stochastic games are zero-sum games (what one player wins the other loses). For every starting node s there is a well-defined value x_s of the game, which is the probability that player 1 wins if both players play optimally. Although the players are allowed to use randomization in each step and to have their choice depend on the entire history of the game, it is known that there are stationary, pure (deterministic) optimal strategies for both players, i.e. strategies where in each step a player takes a deterministic action that does not depend on the past history but only on the current node. Such a strategy σ_i for player $i = 1, 2$ is simply a choice of an outgoing edge (a successor) for each node in V_i; thus there is a finite number of such pure strategies. For every pure strategy profile (σ_1, σ_2) for the two players, the game reduces to a Markov chain $G[\sigma_1, \sigma_2]$, obtained from G by giving probability 1 to all the selected edges out of the player 1 and 2 nodes, and removing the nonselected edges; the values $x_s(\sigma_1, \sigma_2)$ can be computed then by solving a linear system of equations. If the given edge probabilities are rational then the optimal values x_s are also rational, of bit complexity polynomial in the input size. If there are only two of the three types of nodes in the graph, then the optimal strategies and the values x_s can be computed in polynomial time. For example, if there is no player 2, then the game becomes a *Markov decision process* with the goal of maximizing the probability of reaching a sink labeled 1, which can be optimized by Linear Programming. When we have all three types of nodes, the decision problem $x_s \geq 1/2$? (starting from position s, does player 1 win with probability at least 1/2, or any other given rational number $r \in (0, 1)$) is in NP ∩ coNP (and in fact in *UP* ∩ *coUP*). It is a well-known open question whether the problem is in P [13].

Two (pure) strategies σ_1, σ_2 of the two players form an *equilibrium* if σ_1 is a best response of player 1 to the strategy σ_2 of player 2, and vice versa, σ_2 is a best response of player 2 to σ_1. That is, σ_1 is a maximizing strategy in the Markov decision process $G[\sigma_2]$ obtained when the strategy of player 2 is fixed to σ_2, and similarly σ_2 is optimal for player 2 in $G[\sigma_1]$. Note that in fact there is a strategy σ_1 for player 1 that is simultaneously optimal for all starting positions, and similarly for player 2. The equilibria of the game are precisely the optimal strategy pairs.

The problem of computing the values x_s of the game and an equilibrium (a pair of optimal strategies) can be viewed as a local search problem in PLS [14,15]. Let us take the point of view of one player, say player 1: the solution set is the set of pure stationary strategies of player 1, the value assigned to a strategy σ_1 is $\sum_{s\in V} x_s(\sigma_1, \sigma_2)$ where σ_2 is a best response of player 2 to σ_1 (note, σ_2 and hence the value can be computed in polynomial time), and the neighbors of σ_1 are the strategies obtained by switching the choice of a node in V_1. The locally optimal solutions to this PLS problem are the (globally) optimal strategies of player 1. From an optimal strategy σ_1^* of player 1, we can compute in polynomial time an optimal strategy σ_2^* of player 2 and the values x_s of the game for all $s \in V$.

A *mean payoff* game [16] is a non-stochastic two-player game played on a directed graph $G = (V, E)$ with no sinks, whose nodes are partitioned into two sets V_1, V_2 and whose edges are labeled by (rational) rewards $r(e), e \in E$. As above, play starts at a node and moves along the edges, where player 1 chooses the next edge for nodes in V_1 and player 2 for nodes in V_2 (there are no random nodes here), and play goes on forever. The payoff to player 1 from player 2 of a history using the sequence of edges $e_1, e_2, \ldots$ is the average reward per step, $\limsup_{n\to\infty}(\sum_{j=1}^{n} r(e_j))/n$. Again there are optimal pure stationary strategies σ_1, σ_2 for the players, and these form a path followed by a cycle C; the payoff (value of the game) is the ratio $\sum_{e\in C} r(e)/|C|$ and is rational of polynomial bit complexity. As shown in [17], the optimal values and optimal strategies can be computed in pseudopolynomial time (i.e. polynomial time for unary rewards); furthermore, the problem can be reduced to simple stochastic games, it is thus in PLS and the decision problem is in *UP* ∩ *coUP*, but it is open whether it is in P.

A still simpler, nonstochastic two-player game, called the *parity game* [18] has been studied extensively in the verification area; it is an important theoretical question in this area whether this game can be solved in polynomial time. A parity game is played again on a directed graph G whose nodes are partitioned into two sets V_1, V_2 and whose edges are labeled by positive integers. A history is winning for player 1 (respectively player 2) if the maximum label that occurs infinitely often in the history is odd (resp. even). In this game, one of the two players has a pure optimal strategy that wins on every history that results against every strategy of the other player. Determining who the winner of the game is (and a winning strategy) reduces to the decision problem for mean payoff games and in turn to simple stochastic games [19,20].

4. Fixed points

Nash's theorem asserts that every finite game Γ has a (generally, mixed) equilibrium. Nash proved his theorem in [6] using Brouwer's fixed point theorem: every continuous function F from a compact convex body to itself has a fixed point, i.e. a point x such that $x = F(x)$. Specifically, given a finite game Γ with k players $i = 1, \ldots, k$, a finite set S_i of pure strategies and a payoff function U_i for each player, a mixed strategy profile is a vector $x = (x_{ij} | i = 1, \ldots, k; j = 1, \ldots, |S_i|)$, where x_{ij} is the probability with which player i plays pure strategy j. The vector x lies on the product Δ of the k unit simplices $\Delta_i = \{y \in R^{|S_i|} | \sum_{j=1}^{|S_i|} y_j = 1; y \geq 0\}$. Nash

defined the following function from Δ to itself: $F_\Gamma(x)_{(i,j)} \doteq \frac{x_{i,j}+\max\{0,g_{i,j}(x)\}}{1+\sum_{l=1}^{|S_i|}\max\{0,g_{i,l}(x)\}}$, where $g_{i,j}(x)$ is the (positive or negative) 'gain' in payoff of player i if he switches to pure strategy j while the other players continue to play according to x; $g_{i,j}(x)$ is a (multivariate) polynomial in x. Nash showed that the fixed points of F_Γ are precisely the equilibria of the game Γ. There are several alternative proofs of Nash's theorem, all using Brower's theorem (with different functions F) or the related Kakutani's theorem (for fixed points of multivalued maps). Note that the underlying solution space here, Δ, is continuous, not discrete and finite. Furthermore, even if the payoff functions of the game are rational-valued, for three or more players it may be the case that all equilibria are irrational.

Market equilibria are another important application of fixed point theorems. Consider the following exchange model [21]. We have m agents and n commodities. Each agent $l = 1,\ldots,m$ comes to the market with an initial supply (endowment) s^l of commodities, which he exchanges for his preferred commodities. Given a vector p of prices for the commodities, each agent l sells his supply at the prevailing prices, and buys his preferred bundle (demand) $d^l(p)$ of commodities. Thus, for each vector p of prices for the commodities, each agent l has an (positive or negative) 'excess demand' (=demand-supply) $g_i^l(p) = d_i^l(p) - s_i^l$ for each commodity i. Standard assumptions are that the functions $g_i^l(p)$ (i) are homogeneous of degree 0; thus the price vectors may be normalized to lie on the unit simplex Δ_n, (ii) they satisfy Walras' law $\sum_{i=1}^n p_i g_i^l(p) = 0$ (money gained=money spent), and (iii) they are continuous on the unit simplex. Let $g_i(p) = \sum_l g_i^l(p)$ be the (total) market excess demand for each commodity i. The functions $g_i(p)$ satisfy the same constraints. A vector p of prices is an *equilibrium* if $g_i(p) \le 0$ for all i (demand does not exceed supply), with equality for all commodities i that have $p_i > 0$. Brouwer's theorem can be used to show the existence of equilibria. Namely, the equilibria are the fixed points of the function $F : \Delta_n \mapsto \Delta_n$, defined by the formula $F_i(p) = \frac{p_i+\max(0,g_i(p))}{1+\sum_{j=1}^n \max(0,g_j(p))}$. In fact, the equilibrium existence theorem can be conversely used to show Brouwer's theorem: from a Brouwer function one can construct an economy whose equilibria correspond to the fixed points of the function [22].

In the classical Arrow-Debreu market model [23], the user preferences for the commodities are modeled by utility functions, which in turn induce the excess demand functions (or correspondences, i.e. multivalued maps), and more generally the model includes also production. Under suitable conditions, the existence of equilibria is derived again using a fixed point theorem (Kakutani in [23], or Brouwer in alternative proofs [24]). In general, there may be multiple price equilibria, but under certain conditions the equilibrium is unique. As shown in a line of work by Sonnenschein, Mantel and Debreu (see e.g. [25]), essentially any function $g_i(p)$ satisfying the above standard assumptions (i)–(iii) for exchange demand functions can arise actually as the excess demand function in a market for suitably defined utility functions for the users. Thus, there is a tight connection between fixed points of general functions and market equilibria.

A number of other search problems from various domains can be cast as fixed point computation problems in the following sense: every instance I of a problem is associated with a function F_I over some domain so that the sought objects $\mathrm{Sol}(I)$ are fixed points of F_I; in some cases, we may only want a specific fixed point of the function. We will describe several more examples in this section.

Recall the simple stochastic game from the last section. The vector $x = (x_s|s \in V)$ of winning probabilities for player 1 satisfies the following system of equations $x = F(x)$, with one equation for each node s, depending on the type of the node: if s is a sink labeled 1 (respectively 2) then $x_s = 1$ (resp. $x_s = 0$); if $s \in V_r$ then $x_s = \sum_{(s,v)\in E} p_{sv}x_v$; if $s \in V_1$ then $x_s = \max\{x_v|(s,v) \in E\}$; if $s \in V_2$ then $x_s = \min\{x_v|(s,v) \in E\}$. In general there may be multiple solutions to this system of equations; however, the system can be preprocessed so that there is a unique solution in the unit cube $C_n = \{x|0 \le x_s \le 1, \forall s \in V\}$ [13].

The area of stochastic games was initiated by Shapley in his classical paper [26], and the area has grown extensively since then with a rich body of literature; see e.g. [27,28]. Shapley's original games (and many others, including the standard one-shot games) have a more general form, where players can move simultaneously. As shown in [13], simple stochastic games (which are 'turn-based' games, i.e. only one player moves at a time) can be reduced to *Shapley's game*. In *Shapley's game* there is a finite set V of states, each state u has an associated one-shot zero-sum finite game with a reward (payoff) matrix A_u whose rows (resp. columns) correspond to the actions (pure strategies) of player 1 (resp. 2). If the play is in state u, and the players 1, 2 choose respectively actions i, j, then player 1 receives reward $A_u[i,j]$ from player 2, the game stops with probability $q_{ij}^u > 0$, and it transitions to state v with probability p_{ij}^{uv}, where $q_{ij}^u + \sum_v p_{ij}^{uv} = 1$. The input to the Shapley game consists of the set of states V, the matrices $A_u, u \in V$, and all the stopping probabilities q_{ij}^u, and transition probabilities p_{ij}^{uv} for all states and pairs of actions. For computational purposes we assume as usual that all the input data (i.e., all rewards and probabilities) are rational.

The game starts at a given state s and then proceeds in each step from state to state according to the actions chosen by the player and the corresponding probabilistic outcomes. Since there is at least positive probability $q = \min\{q_{ij}^u|u,i,j\} > 0$ of stopping in each step, the game stops almost surely (i.e., with probability 1) in a finite number of steps. The goal of player 1 is to maximize (and of player 2 to minimize) the total expected reward, which is the value of the game.[1] Both players have optimal stationary strategies, i.e., strategies that depend in each step on the current state but not on the past history, but the optimal strategies are mixed (randomized) in

[1] Another standard equivalent formulation of a Shapley game is as a discounted game, where the game does not stop but future rewards are discounted by a factor $1-q$ per step. That is, in the discounted formulation there are no stopping probabilities q_{ij}^u and $\sum_v p_{ij}^{uv} = 1$ for all u,i,j. There is instead a discount factor $q > 0$ specified, and the objective of player 1 (respectively, player 2) is to maximize (resp. minimize) the expectation of the discounted aggregate reward $\sum_{t=0}^\infty (1-q)^t r(t)$, where $r(t)$ is the reward in step t.

this case. Note that this is true even in the case of one-shot zero-sum games, which is the special case of a Shapley game where there is only one state and the stopping probability is 1.

Given a Shapley game G, we want to compute the vector $x = (x_u | u \in V)$ of game values for the different starting states u. The values in general may now, however, be irrational. The vector x satisfies a fixed point set of equations $x = F(x)$, which is constructed as follows. For each state u, let $B_u(x)$ be the matrix, whose rows and columns are indexed by the actions of the players, and whose i, j entry is $B_u[i, j] = A_u[i, j] + \sum_v p_{ij}^{uv} x_v$. Let $\mathrm{Val}(B_u(x))$ be the value of the one-shot zero-sum game with payoff matrix $B_u(x)$. Then $x = F(x)$, where $F_u(x) = \mathrm{Val}(B_u(x)),\ u \in V$. The function F is a Banach function (a contraction map) under the L_∞ norm with contraction factor $1 - q$; that is, for any two vectors x, y, the function F satisfies $|F(x) - F(y)| \leq (1 - q)|x - y|$. It follows from Banach's theorem that F has a unique fixed point: the vector of values of the game.

Branching processes are a basic model of stochastic evolution, introduced first in the single type case by Galton and Watson in the 19th century to study population dynamics, and extended later to the multitype case by Kolmogorov and Sevastyanov, motivated by biology [70]. A branching process has a finite set T of n types, for each type $i \in T$ there is a finite set of 'reproduction' rules of the form $i \xrightarrow{p_{ij}} v_{ij}, j = 1, \ldots, m_i$, where $p_{ij} \in (0, 1]$ is the probability of the rule (thus, $\sum_{j=1}^{m_i} p_{ij} = 1$) and $v_{ij} \in \mathbb{N}^n$ is a vector whose components specify the number of offsprings of each type that an entity of type i produces in the next generation. Starting from an initial population, the process evolves from one generation to the next according to the probabilistic reproduction rules. The basic quantity of interest is the probability x_i of extinction of each type $i \in T$: the probability that if we start with one individual of type i, the process will eventually die. These probabilities x_i can be used to compute the extinction probability for any initial population and are the basis for more detailed statistics of the process. As usual, we assume that the probabilities of the rules are rational. However, the extinction probabilities are in general irrational. The vector x of extinction probabilities satisfies a set of fixed point equations $x = F(x)$, where $F_i(x) = \sum_{j=1}^{m_i} p_{ij} \prod_{k=1}^{n} (x_k)^{v_{ij}[k]}$. Note that each $F_i(x)$ is a (multivariate) polynomial with positive coefficients; thus F is a monotone operator on $\mathbb{R}^n_{\geq 0}$, and therefore it has a Least Fixed Point (LFP), i.e. there is a vector $x^* \in \mathbb{R}^n_{\geq 0}$ such that (i) $x^* = F(x^*)$, and (ii) $x^* \leq y$ for any other vector $y \in \mathbb{R}^n_{\geq 0}$ such that $y = F(y)$. The LFP x^* is precisely the vector of extinction probabilities of the branching process. For more information on the theory of branching processes and their applications, see [29–31].

Stochastic context-free grammars (SCFGs) are context-free grammars where the production rules have associated probabilities. They have been studied extensively in Natural Language Processing where they are an important model [32], and have been used also in biological sequence analysis. An SCFG $G = (T, V, R, S_1)$ has a finite set T of terminals, a finite set $V = \{S_1, \ldots, S_n\}$ of nonterminals (variables), where S_1 is the starting nonterminal, and a finite set R of probabilistic production rules for the nonterminals of the form $S_i \xrightarrow{p_{ij}} \alpha_{ij}, j = 1, \ldots, m_i$, where $\alpha_{ij} \in (T \cup V)^*$ is a string and the rule probabilities $p_{ij} \in (0, 1]$ satisfy $\sum_{j=1}^{m_i} p_{ij} = 1$. An SCFG G generates a language $L(G) \subseteq T^*$ and associates a probability $p(\tau)$ to every terminal string τ in the language, according to the following stochastic process. The process starts with the starting nonterminal S_1 and derives (probabilistically) a sequence of strings $\sigma_0 = S_1, \sigma_1, \sigma_2, \ldots$ over $T \cup V$, where in each step t we replace the leftmost nonterminal, say S_i, in the current string σ_t with the string α_{ij} on the right-hand side of a rule for S_i, which is chosen probabilistically with probability p_{ij}. The process terminates when (and if) the current string has no nonterminals. The probability $p(\tau)$ of a terminal string is the probability that the process terminates with the string τ. The probability of the language $L(G)$ of the SCFG G is $p(L(G)) = \sum_{\tau \in L(G)} p(\tau)$. (The leftmost derivation policy above is arbitrary: all policies yield the same probabilities.) The probability $L(G)$ again may be an irrational number even if all the probabilities of the production rules are rational. The analysis of SCFGs is closely related to that of branching processes. Note that $L(G)$ is the probability that the stochastic process that we described above, starting with S_1, terminates. More generally, we can define for each nonterminal $S_i \in V$ an associated probability $p(S_i)$, which is the probability that the process starting with S_i terminates. If x is a vector of variables corresponding to the probabilities $p(S_i)$, we can form a set of equations $x = F(x)$, where each $F_i(x)$ is a polynomial with positive coefficients, such that the LFP x^* of F is the vector of probabilities $p(S_i)$; the first entry x^*_1 is $L(G)$.

A model that encompasses and generalizes SCFGs and extinction probabilities of branching processes (and other models studied in different areas, e.g. *quasi-birth–death processes*, studied in queuing theory and performance analysis, see e.g. [33] and the references in [34]; backbutton processes for web surfing [35]) is the *Recursive Markov chain* (RMC) model [36] and the equivalent model of *Probabilistic Pushdown machines* [37]. Informally, an RMC is a collection of Markov chains that can call each other in a potentially recursive manner like recursive procedures. The basic quantities of interest are the termination and reachability probabilities: If the RMC starts at a given node, what is the probability that it terminates? What is the probability that it terminates at a particular exit (sink)? These probabilities form the basis for more general analysis of RMCs. The termination probabilities obey again a system of fixed point equations $x = F(x)$, where F is a vector of polynomials with positive coefficients; the least fixed point of the system gives the termination probabilities of the RMC for different starting and ending nodes. Generalization to a setting where the dynamics are not completely probabilistic but can be controlled by one or more players leads to *recursive Markov decision processes and games* [38–40]. For example, we may have a branching process, where the reproduction of some types can be influenced by players who want to bias the process towards extinction or towards survival. This results in fixed point systems of equations involving monotone polynomials and the min and max operators.

All of the above problems are total (single-valued or multi-valued) search problems, in which the underlying solution space is continuous. In all of these problems we would ideally like to compute exactly the quantities of interest if possible (if they are rational), and otherwise, i.e. if they are irrational,

which is the case with many of these problems, we would like to address the associated discrete problems: (1) Bound the quantities and answer decision questions about them; for example: Is the value of a stochastic game $\geq 1/2$? Does an RMC terminate with probability 1? (2) Approximate the quantities within desired precision, i.e. compute a solution x that is within ϵ of an/the answer x^* to the search problem; for example, approximate within additive error ϵ the extinction probabilities of a branching process, compute a mixed strategy profile for a game that is within ϵ of a Nash equilibrium, and estimate the prices in a market equilibrium (*the* equilibrium if there is a unique one). In the approximation problem we would like ideally polynomial time in the size of the input and in $\log(1/\epsilon)$ (the number of bits of precision).

We refer to the approximation of an (actual) solution to a search problem as above as *strong* approximation (or the 'near' problem) to distinguish it from another notion of approximation, which we call the *weak* approximation (or the 'almost' problem) that is specific to a fixed point formulation of a search problem via a function F: a weak ϵ-approximation is a point x such that $|x - F(x)| \leq \epsilon$ (say in the L_∞ norm). Note that a search problem may be expressible in different ways as a fixed point problem using different functions F, and the notion of weak approximation may depend on the function that is used. The (strong) approximation notion concerns the approximation of the actual solutions to the search problem, i.e., it is intrinsic to the problem itself and does not depend on F.

For many common fixed point problems (formally, for polynomially continuous functions [41]), including all of the above problems, weak approximation reduces to strong, i.e., given instance I and (rational) $\epsilon > 0$, we can define a (rational) $\epsilon' > 0$ of bit-size polynomial in that of ϵ and in $|I|$ such that every (strong) ϵ'-approximation x to a solution to the search problem (i.e., approximation to a fixed point of the function F_I corresponding to the instance I) is a weak ϵ-approximate fixed point (i.e., satisfies $|x - F_I(x)| \leq \epsilon$). The converse relation does not hold in general; for example, it does not hold for Nash equilibria and the Nash function F_Γ. Points that are sufficiently close to actual fixed points (equilibria) are weakly ϵ-approximate, but there may well be other points that are weakly ϵ-approximate and which are far away from all the equilibria. This is shown schematically in Fig. 1. The analogous situation in local search is that a solution may be approximately optimal in its local neighborhood (i.e., none of the neighboring solutions is significantly better), but it may be far (in both distance and value) from all local optima; a move with a small improvement may then trigger a long sequence of improvements that ends up in another part of the space.

For some problems, finding weakly approximate fixed points is also useful and has a natural meaning. For many problems, however, the function F_I is just a tool that allows us to prove existence of the desired object and/or to express it mathematically in terms of a system of equations, and a weakly approximate fixed point (i.e. a point that approximately satisfies the equations) may have no relevance. For example, if we are interested in the probability of some event in a stochastic model or the value of a game, then a point x that satisfies $|x - F(x)| \leq \epsilon$, but which is far from the desired probability or value, is not useful. Indeed, for several models we can easily compute such weakly ϵ-approximate solutions for every constant $\epsilon > 0$ (see below), but it is much harder (and open) to approximate the actual desired quantities (probabilities, values) within a nontrivial constant.

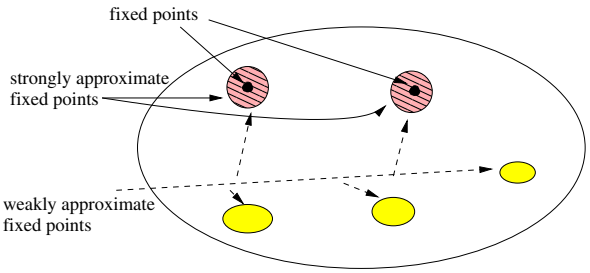

Fig. 1 – Approximate fixed points.

We now briefly discuss algorithms for fixed point problems. If F_I is a Banach function, i.e., a contraction map with a contraction factor $1 - q$, we can start at any point x_0, and repeatedly apply F_I. The process will converge to the unique fixed point. If the contraction factor $1 - q$ is a constant < 1, then convergence is polynomial, but if the margin q from 1 is very small, inverse exponential in the size of the input I (as is generally the case, for example, in *Shapley's game*), then convergence is slow.

If F_I is a monotone function for which we want to compute the least fixed point x^*, as in many of the examples above (stochastic games, branching processes, RMC, etc.), we can start from $x_0 = 0$ (a point that does not exceed the LFP) and repeatedly apply F_I; the process will converge to the desired LFP x^*, but again convergence is generally slow. Note that, for many of these problems, obtaining a weak ϵ-approximation for ϵ constant or even inverse polynomial, $|I|^{-c}$, is easy: for example, in a simple stochastic game (or a branching process, an RMC, etc.), the vector x is bounded from above by the all-1 vector and the iterates $F_I^k(0), k = 0, 1, 2, \ldots$ increase monotonically with k. If every iteration increases at least one coordinate by at least ϵ, then we cannot have more than n/ϵ iterations. Thus, after at most n/ϵ iterations we will get a weak ϵ-approximate fixed point x. However, such a point x is of no use in estimating the actual values or probabilities that we want to compute. Approximating the value of a simple stochastic game even within additive error $1/2$ is an open problem [13].

For general Brouwer functions F we cannot simply iteratively apply F from some starting point x_0 and hope to converge to a fixed point. There is extensive algorithmic work on the approximate computation of Brouwer fixed points, starting with Scarf's fundamental algorithm [42], The standard proof of Brouwer's theorem involves a combinatorial lemma, Sperner's lemma, combined with a (generally nonconstructive) compactness argument. Scarf's algorithm solves Sperner's problem constructively, and computes a weak ϵ-approximate fixed point for the function. Briefly, it works as follows. Assume without loss of generality that the domain is the unit simplex $\Delta_n = \{x \geq 0 | \sum_i x_i = 1\}$, and consider a simplicial subdivision of Δ_n into simplices of

sufficiently small diameter δ, so that $|x-y| \leq \delta$ implies $|F(x) - F(y)| \leq \epsilon/n$. Label ('color') each vertex v of the subdivision by an index $i = 1, \ldots, n$ such that $v_i > F_i(v)$; if v is not a fixed point there is at least one such index; if v is a fixed point then label v with, say, $\arg_i \max(v_i)$. Note that the unit vectors e_i at the n corners of the simplex Δ_n are labeled i, and all vertices on the facet $x_j = 0$ are labeled with an index $\neq j$. Sperner's lemma implies then that the subdivision has at least one panchromatic simplex, i.e. a small simplex S whose vertices have distinct labels. From the definition of the labels and the choice of δ it follows that any point $x \in S$ satisfies $|F(x) - x| \leq \epsilon$. Scarf's algorithm starts with a suitable subdivision and a boundary simplex whose vertices have $n-1$ distinct indices (all except one), and then keeps moving to an adjacent simplex through the face with the $n-1$ indices; the process cannot repeat any simplex of the subdivision, so it will end up at a panchromatic simplex S. Note that S may not contain any actual fixed points, and in fact may be located far from all of them, but any point x of S is a weak ϵ-approximation. If we take finer and finer subdivisions letting the diameter δ go down to 0, then the resulting sequence of weakly approximate fixed points must contain (by compactness) a subsequence that converges to a point, which must be a fixed point; this latter part, however, is nonconstructive in general.

There are several other subsequent methods for computing (approximate) fixed points, e.g. Newton-based, and homotopy methods (some of these assume differentiability and use also the derivatives of the function). Scarf's algorithm, as well other general-purpose algorithms, treat the function F as a black box. Such black box algorithms must take exponential time in the worst case to compute a weak approximation [43, 67]. Furthermore, for strong approximation no finite amount of time is enough in the black box model [44], and there are also noncomputability results for computing equilibria and fixed points for a model where the function is given via a Turing machine [45,46]. However, the restriction to black box access is a severe one, and the results do not mean that any of the specific problems we want to solve (for example, Nash equilibria) is necessarily hard.

5. Rational equilibria, piecewise linear functions and the class PPAD

Consider a two-player finite game, with the payoffs given explicitly in terms of the two payoff matrices A_1, A_2 of the two players, where the rows of the matrices correspond to the pure strategies of player 1 and the columns to those of player 2 (i.e., the game is presented in *normal form*). We assume as usual that the payoffs are rational for computational purposes. The sets of mixed strategies M_1, M_2 of the two players are the probability distributions on the rows and columns, and can be represented by vectors of appropriate dimensions. In the case of *zero sum* games, i.e., when $A_1 + A_2 = 0$, the game has a well-defined value $v^* = \sup_{x \in M_1} \inf_{y \in M_2} x^T A_1 y = \inf_{y \in M_2} \sup_{x \in M_1} x^T A_1 y = -\sup_{y \in M_2} \inf_{x \in M_1} x^T A_2 y$ by Von Neuman's minimax theorem; a Nash equilibrium consists of a pair of optimal strategies for the players, and the value and optimal strategies can be computed efficiently using Linear Programming [47]. For general payoff matrices A_1, A_2, there may be several different Nash equilibria with different payoffs. Computing a specific Nash equilibrium, such as one that maximizes the payoff to one of the players, or to all the players, is NP-hard [48]. However, the search problem that asks for *any* Nash equilibrium is a different, potentially 'easier' problem, and is unlikely to be NP-hard.

The two-player case of the Nash equilibrium problem can be viewed either as a continuous or as a discrete problem, like Linear Programming: We can consider LP either as having a continuous solution space, namely all the real-valued points in the feasible polyhedron, or as having a discrete solution space, namely the vertices of the polyhedron or the feasible bases. The same holds, for two-player games which correspond to a Linear Complementarity problem. A mixed strategy profile is a Nash equilibrium iff every pure strategy of each player is either at 0 level (not in the support) or is a best response to the strategy of the other player. Assuming the game is nondegenerate (we can always ensure this by a small perturbation) the supports of the mixed strategies determine uniquely the equilibrium: we can set up and solve a linear system of equations which equate the payoffs of the pure strategies in the support of each player, and check that the solution satisfies the appropriate inequalities for the pure strategies that are not in the supports.

One consequence of this is that if the payoffs are rational then there are rational equilibria, of polynomial bit complexity in the input size, and they can be computed exactly. A second consequence is that Nash's theorem in this case can be proved directly, without resorting to a fixed point theorem, and algorithmically, namely by the Lemke–Howson algorithm [49]. The algorithm has similar flavor to Scarf's algorithm for fixed points. Mixed profiles can be labeled ('colored') by the set of pure strategies that are not in the support or that are best responses to the other player's strategy. The equilibria are the mixed profiles that are panchromatic, i.e., labeled with all the pure strategies of both players. Briefly, the algorithm starts from an artificial point that has all the colors except one, and then follows a path through a sequence of LP-like pivots, until it arrives at a panchromatic point (profile), which must be an equilibrium; the algorithm cannot repeat any point, because at any point there are only two possible pivots, one forward and one backward, and there is a finite number of points (supports), so it terminates. It is known that the algorithm takes exponential time in the worst case [50].

Papadimitriou defined in [51] a complexity class, PPAD, that captures the basic principles of these path-following algorithms: There is a finite number of candidate solutions, and an underlying directed graph of moves between the solutions where each solution has at most one forward and one backward move, i.e., the graph consists of a set of directed paths, cycles and isolated nodes; a source of one path is an artificial starting solution, and every other endpoint (source or sink) of every path is an answer to the problem (e.g., an equilibrium).

Formally, a search problem Π is in PPAD if each instance I has a set $S(I)$ of candidate solutions which are (strings) polynomially bounded in the input size $|I|$, and there are

polynomial-time algorithms for the following tasks: (a) test whether a given string I is an instance of Π and if so compute a initial candidate solution s_0 in $S(I)$, (b) given I, s, test whether $s \in S(I)$ and if so compute a successor $\text{succ}_I(s) \in S(I)$ and a predecessor $\text{pred}_I(s) \in S(I)$, such that $\text{pred}_I(s_0) = s_0$, $\text{succ}_I(s_0) \neq s_0$, and $\text{pred}_I(\text{succ}_I(s_0)) = s_0$. The *pred* and *succ* functions induce a directed graph $G = (S(I), E)$, where $E = \{(u, v) | u \neq v, \text{succ}_I(u) = v, \text{pred}_I(v) = u\}$, and the desired solution set to the instance I of the search problem, $\text{Sol}(I)$, is the set of nodes of G, other than s_0, that have indegree + outdegree = 1, i.e., are endpoints of the paths; note that $\text{Sol}(I) \neq \emptyset$ because there must be at least one more endpoint besides s_0. As is customary, the class is closed under polynomial-time reductions, i.e., if a search problem A reduces to a problem B that satisfies the above definition, then A is considered also to belong to PPAD. Papadimitriou defined two other variants of this class in [51]: PPA, in which the underlying graph is undirected, and PPADS, in which the graph is directed and the desired solution set $\text{Sol}(I)$ consists only of the sinks of the paths. However, PPAD is the more interesting and richer of these classes in terms of natural problems.

The class PPAD lies somewhere between P and TFNP: all search problems in PPAD are total, and furthermore, for a given instance I, we can guess a solution $s \neq s_0$ and verify that it is a source or sink in the graph, i.e. that $s \in \text{Sol}(I)$. Thus, as in the case of PLS, problems in PPAD cannot be NP-hard unless NP = coNP.

By virtue of the Lemke–Howson algorithm, the Nash equilibrium problem for two-player (normal form) games is in PPAD. For three or more players we cannot say that the Nash problem is in PPAD; for one thing the equilibria are irrational. But the following approximate ϵ-Nash version is in PPAD [52]. An *ϵ-Nash equilibrium* of a game is a (mixed) strategy profile such that no player can improve its payoff by more than ϵ by switching unilaterally to another strategy. (Note, this is not the same as being ϵ-close to a Nash equilibrium.) The ϵ-Nash problem is: Given a normal form game Γ (with rational payoffs) and a rational $\epsilon > 0$, compute an ϵ-Nash equilibrium of Γ. (Note that ϵ is given as usual in binary, so polynomial time means polynomial in $|\Gamma|$ and $\log(1/\epsilon)$.)

The complexity of the Nash problem was one of the main motivations for the original introduction of PPAD. A recent sequence of papers culminated in showing that the Nash equilibrium problem for two-player games is PPAD-complete [52,53]; that is, if the problem can be solved in polynomial time, then so can all the problems in PPAD. Furthermore, even the ϵ-Nash equilibrium problem for ϵ inverse polynomial, i.e. even with ϵ given in unary, is also PPAD-complete for two-player games [54], and even if all the payoffs are 0 or 1 ('win–lose' games) [55]. For all constant ϵ, an ϵ-Nash equilibrium can be computed in quasipolynomial time, specifically in time $n^{O(\log n/\epsilon^2)}$, for any fixed number of players [56]. It is open whether there is a polynomial time (additive) approximation scheme, even for two players. There are several papers that give polynomial-time algorithms for some constant ϵ (e.g. [68,71]); see [57] for an overview of the progress so far on the constant ϵ for which ϵ-Nash (and the related concept of *well-supported* ϵ-Nash) equilibria can be computed in polynomial time in two-player games.

Another basic PPAD-complete problem is (a formalization of) the Sperner problem. The two-dimensional (2D) case concerns the unit simplex (triangle) Δ_3 and its simplicial subdivision (i.e., triangulation) with vertices $v = (i_1/n, i_2/n, i_3/n)$ with $i_1 + i_2 + i_3 = n$. The 2D Sperner problem is as follows. The input consists of a number n in binary and a Boolean circuit which takes as input three natural numbers i_1, i_2, i_3 with $i_1 + i_2 + i_3 = n$ and outputs a color $c \in \{1, 2, 3\}$, with the restriction that $i_c \neq 0$. The problem is to find a trichromatic triangle, i.e. three vertices (triples) with pairwise distances $1/n$ that have distinct colors. The problem is in PPAD by Scarf's algorithm. The 3D Sperner problem was shown to be PPAD-complete in [51], and the 2D case was shown to be complete in [58].

The original paper of Papadimitriou [51] showed PPAD-completeness also for a discretized version of Brouwer and related theorems (Kakutani, Borsuk–Ulam) in the style of the Sperner problem: a Brouwer function in three dimensions is given in terms of a binary number n (the resolution of a regular grid subdivision of the unit 3-cube) and a Boolean circuit that takes as input three natural numbers i_1, i_2, i_3 between 0 and n and outputs the value of the function at the point $(i_1/n, i_2/n, i_3/n)$. The function is then linearly interpolated in the rest of the unit cube according to a standard simplicial subdivision with the grid points as vertices. The paper showed also completeness for a discretized version of the market equilibrium problem for an exchange economy.

In [59], Codenoti et al. show PPAD-completeness of the price equilibrium for a restricted case of Leontief exchange economies, i.e. economies in which each agent i wants commodities in proportion to a specified (nonnegative) vector $(a_{i1}, \ldots, a_{ik})$; that is, the utility function of agent i is $u_i(x) = \min\{x_{ij}/a_{ij} | j = 1, \ldots, k; a_{ij} \neq 0\}$. In general, Leontieff economies may not have an equilibrium. Furthermore, it is NP-hard to determine if there is an equilibrium [59], and even if there exist equilibria, it may be that they all are irrational. However, [59] defines a restricted subclass of Leontief economies that have equilibria and for which the problem is equivalent to the Nash equilibrium problem for two-player games. This restricted Leontief subclass is as follows: the agents are partitioned into two groups, every agent brings one distinct commodity to the market, and agents in the first group want commodities only of agents in the second group and vice versa.

The class PPAD cannot of course capture general Brouwer functions since many of them have irrational fixed points, as we saw in the last section.[2] However, there is a natural class of functions that are guaranteed to have rational fixed points, which are in PPAD and in a sense characterize the class [41]. Consider the search problem Π of computing a

[2] We could discretize such a function F, for example by superimposing a very fine grid, and approximating it by another function G that is guaranteed to have rational fixed points; however, the resulting approximating function G will have new fixed points in general, which may have no relation and can be very far from the fixed points of the original function F.

fixed point for a family of Brouwer functions $\mathcal{F} = \{F_I | I$ an instance of $\Pi\}$. We say that Π is a *polynomial piecewise linear* problem if the following hold: For each instance I, the domain is divided by hyperplanes into polyhedral cells, the function F_I is linear in each cell, and is of course continuous over the whole domain. The coefficients of the function in each cell and of the dividing hyperplanes are rationals of size bounded by a polynomial in $|I|$. These are not given explicitly in the input; in fact, there may be exponentially many dividing hyperplanes and cells. Rather, there is an oracle algorithm that locates the cell of a given point x in time polynomial in $|I|$ by generating a sequence of queries of the form $ax \leq b$? adaptively (i.e., the next query depends on I and the sequence of previous answers), and at the end the algorithm either outputs 'No' (i.e., x is not in the domain) or identifies the cell of x and outputs the coefficients c, c' of the function $F_I(x) = cx + c'$. As an example, piecewise linear functions F_I include those where every coordinate function is given by a piecewise linear formula on the variables, such as $\min(3x_1 + 4x_2, 2x_2 + 1)$, $|x_1 - .2| + \max(x_2, x_3)$, etc. For example, the function F_I for a simple stochastic game described in the previous section is obviously piecewise linear. As shown in [41], all polynomial piecewise linear problems are in PPAD (they all have rational fixed points of polynomial size). Examples include the simple stochastic games, the discretized Brouwer functions obtained from linear interpolation on a grid, and the Nash equilibrium problem for two-player games (Nash's function in nonlinear even for two players, but there is another piecewise linear function whose fixed points are also exactly the Nash equilibria [41]).

The class PPAD captures also the approximation in the weak ('almost') sense for a broad class of Brouwer functions; in some cases this suffices also for the strong approximation ('near') problem [41]. Consider a family of functions $\mathcal{F} = \{F_I\}$. We say $\mathcal{F}$ is *polynomially computable* if, for every instance I and rational vector x in the domain, the image $F_I(x)$ is rational and can be computed in time polynomial in the size of $I$and of x. $\mathcal{F}$ is called *polynomially continuous* if there is a polynomial $q(z)$ such that, for all instances I and all rational $\epsilon > 0$, there is a rational $\delta > 0$ such that $size(\delta) \leq q(|I| + size(\epsilon))$ and such that, for all $x, y \in D_I$, $|x - y| < \delta \Rightarrow |F_I(x) - F_I(y)| < \epsilon$. If $\mathcal{F}$ is polynomially computable and polynomially continuous, then the weak approximation problem (given instance I and rational $\epsilon > 0$, compute a weakly ϵ-approximate fixed point of F_I) can be placed in PPAD using Scarf's algorithm. Furthermore, if the functions F_I happen to be also contracting with contraction rate $< 1 - 2^{-poly(|I|)}$, then strong approximation reduces to weak approximation, and the strong approximation problem (given I, ϵ, compute a point x that is within ϵ of some fixed point x^* of F_I) is also in PPAD; Shapley's problem is an example that satisfies this condition. Moreover, if in addition the functions F_I have rational fixed points of polynomial size, then strongly ϵ-approximate fixed points with small enough ϵ can be rounded to get exact fixed points, and thus the exact problem is in PPAD; simple stochastic games, perturbed with a small discount [13], are such an example.

6. Irrational equilibria, nonlinear functions, and the class FIXP

Games with three or more players are quite different than two-player games in several respects: Nash equilibria are generally irrational; knowing the support of an equilibrium does not help us much in computing one, and there may be many different equilibria with the same support. There are many search problems as we saw in Section 4, and in particular many problems that can be cast in a fixed point framework, where the objects that we want to compute (the solutions) are irrational. Of course we cannot compute them exactly in the usual Turing machine model of computation. As explained in Section 2, we want to compute desired (finite) information about the solutions, giving rise to some basic discrete problems associated with the search problem.

Consider for example Shapley's stochastic game. Some relevant questions about the value of the game are the following: (i) *Decision problem*: Given game Γ and rational r, is the value of the game $\geq r$?, (ii) *Partial computation*: Given Γ, integer k, compute the k most significant bits of the value, (iii) *Approximation*: Given Γ, rational $\epsilon > 0$, compute an ϵ approximation to the value. Similar questions can be posed about the optimal strategies of the players. In general, these problems may not have the same complexity. This is the case in fact for Shapley. As we said in the previous section, the approximation problem for the value of *Shapley's game* is in PPAD (and it is open whether it is P). The decision (and partial computation) problem however seems to be harder and it is not at all clear that it is even in NP; in fact showing that it is in NP would answer a well-known long-standing open problem. The same applies to many other problems. The best upper bound we know for the decision (and partial computation) problem for *Shapley's games* is PSPACE. The same holds for many of the other fixed point problems listed in Section 3 (e.g., branching processes, RMCs, etc), including their approximation version.

The complexity of many problems with continuous domains can be connected to some intriguing open problems in numerical computation: the square root sum problem, and more generally, the power of unit-cost exact rational arithmetic. The *Square Root Sum* problem (`Sqrt-Sum` for short) is the following problem: given positive integers $d_1, \ldots, d_n$ and k, decide whether $\sum_{i=1}^{n} \sqrt{d_i} \leq k$. This problem arises often for example in geometric computations, where the square root sum represents the sum of Euclidean distances between given pairs of points with integer (or rational) coordinates; for example, determining whether the length of a specific spanning tree, or a TSP tour of given points on the plane, is bounded by a given threshold k amounts to answering such a problem. This problem is solvable in PSPACE, but it has been a major open problem since the 1970's (see, e.g., [60,61,73]) whether it is solvable even in NP (or better yet, in P).

A related, and in a sense more powerful and fundamental, problem is the *PosSLP* (Positive Straight-Line-Program) problem: given a division-free straight-line program, or equivalently, an arithmetic circuit with operations $+, -, *$ and inputs 0 and 1, and a designated output gate, determine whether the integer N that is the output of the circuit is positive. The importance of this problem

was highlighted in [62], which showed that it is the key problem in understanding the computational power of the Blum–Shub–Smale model of real computation [1] using rational numbers as constants, in which all operations on rationals take unit time, regardless of their size. Importantly, integer division (the floor function) is not allowed; unit cost models with integer division or logical bit operations can solve in polynomial time all PSPACE problems; see e.g. [63] for an overview of machine models and references. The unit cost rational arithmetic model (without integer division) is a powerful model in which the Sqrt-Sum problem can be decided in polynomial time [61]). Allender et al. [62] showed that the set of discrete decision problems that can be solved in P-time in this model is equal to P^{PosSLP}, i.e. problems solvable in P using a subroutine for PosSLP. They showed also that PosSLP and Sqrt-Sum lie in the Counting Hierarchy (a hierarchy above PP).

The Sqrt-Sum problem can be reduced to the decision version of many problems: the Shapley problem [41], concurrent reachability games [39], branching processes, Recursive Markov chains [36], and Nash equilibria for three or more players [41]. The PosSLP problem also reduces to several of these. Hence placing any of these problems in NP would imply the same for Sqrt-Sum and/or PosSLP. Furthermore, for several problems, the approximation of the desired objects is also at least as hard. In particular, approximating the termination probability of a Recursive Markov chain within any constant additive error $< 1/2$ is at least as hard as the Sqrt-Sum and the PosSLP problems [41].

Similar results hold for the approximation of Nash equilibria in games with three or more players. Suppose we want to estimate the probability with which a particular pure strategy, say strategy 1 of player 1, is played in a Nash equilibrium (any one); obviously, the value 1/2 estimates it trivially with error $\leq 1/2$. Guarranteeing a constant error $< 1/2$ is at least as hard as the Sqrt-Sum and the PosSLP problems [41], i.e. it is hard to tell whether the strategy will be played with probability very close to 0 or 1. This holds even for games that have a unique Nash equilibrium.

The constructions also illustrate the difference between strong and weak approximate fixed points generally, and for specific problems in particular. Recall that for Recursive Markov Chains (RMCs) we can compute very easily a weak ϵ-approximate fixed point for any constant $\epsilon > 0$; however, it is apparently much harder to obtain a strong approximation, i.e. approximate the actual probabilities within any nontrivial constant. In the RMC case the weak approximation is irrelevant. However, in the case of Nash equilibria, the weak approximation of Nash's function is also natural and meaningful: it is essentially equivalent to the notion of ϵ-Nash equilibrium (there is a small polynomial change in ϵ in each direction). All strategy profiles that are close to an actual Nash equilibrium are also ϵ-Nash; however, in the reverse direction there can be a big gap, i.e. ϵ-Nash equilibria can be far from all actual equilibria. Formally, for every game Γ and $\epsilon > 0$, we can choose a ϵ' of bit-size polynomial in the size of Γ and ϵ so that every strategy profile that is within distance ϵ' of a Nash equilibrium is ϵ-Nash (i.e. all strongly approximate points are also weakly approximate with a 'small' change in ϵ). However, the converse is not true: For every n there is a three-player game of size $O(n)$, with an ϵ-Nash equilibrium, x, where $\epsilon = 1/2^{2^{\Omega(n^c)}}$, such that x has distance $1 - 2^{-poly(n)}$ (i.e., almost 1) from every Nash equilibrium [41].

For two-player games, as we said, there is a direct, algorithmic proof of the existence of Nash equilibria (by the Lemke–Howson algorithm). But for three and more players, the only proofs known are through a fixpoint theorem (and there are several proofs known using different Brouwer functions or Kakutani's theorem). In [41] we defined a class of search problems, FIXP, that can be cast as fixed point problems of functions that use the usual algebraic operations and max, min, like Nash's function, and the other functions for the problems discussed in Section 4. Specifically, FIXP is the class of search problems Π, such that there is a polynomial-time algorithm which, given an instance I, constructs an algebraic circuit (straight-line program) C_I over the basis $\{+, *, -, /, \max, \min\}$, with rational constants, that defines a continuous function F_I from a domain to itself (for simplicity, standardized to be the unit cube; other domains can be embedded into it), with the property that $Sol_\Pi(I)$ is the set of fixed points of F_I. The class is closed as usual under reductions. Since the solutions here are generally real-valued, we use SL-reductions as explained in Section 2, i.e. a reduction from problem A to problem B consists of a polynomial-time computable function f that maps instances I of A to instances $f(I)$ of B, and a second function g that is a separable linear transformation that maps solutions y of the instance $f(I)$ of B to solutions x of the instance I of A.

Examples of problems in FIXP include: Nash equilibrium for normal form games with any number of players, price equilibrium in exchange economies with excess demand functions given by algebraic formulas or circuits, the value (and optimal strategies) for Shapley's stochastic games, extinction probabilities of branching processes, and probability of languages generated by stochastic context-free grammars.

FIXP is a class of search problems with continuous solution spaces, and corresponding to each such problem Π, there are the associated discrete problems: decision, approximation, etc. (Formally, one can define corresponding classes FIXP_d, FIXP_a for the discrete problems associated with problems in FIXP.) All the associated discrete problems can be expressed in the existential theory of the reals, and thus, using decision procedures for this theory [64,65], it follows that they are all in PSPACE. As we mentioned, many of these problems are at least as hard as the Sqrt-Sum and the PosSLP problems, for which the current best upper bounds are barely below PSPACE. On the other hand, we do not know of any lower bounds, such as NP-hardness results, so in principle they could all be in P, though this is extremely doubtful; it would mean for one thing that one can simulate unit cost exact rational arithmetic in polynomial time.

The Nash equilibrium problem for three players is complete for FIXP. It is complete in all senses, e.g., its approximation problem is as hard as the approximation of any other FIXP problem, the decision problem is at least as hard as the decision problem for any problem in FIXP, etc. The price equilibrium problem for algebraic excess demand functions is another complete problem.

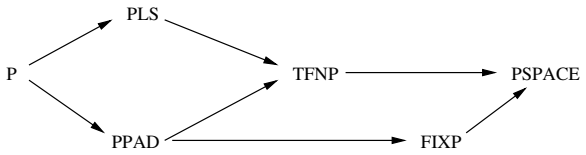

Fig. 2 – Relations between the complexity classes.

The class FIXP is robust; it remains the same under several variations in the definition, as a consequence of reductions and the completeness results. For example, the domain of the function can be any polyhedron, constructible in polynomial time from the input, or a nonlinear domain such as an ellipsoid. Adding roots and fractional powers to the basis does not change the class. In the other direction, we can restrict the function to be represented using formulas instead of circuits; this does not affect the class because Nash's function is given by a formula. Also, FIXP stays the same if we use circuits over $\{+, *, \max\}$ only and rational constants (i.e., no division), because there is another function for the Nash problem whose fixed points are also the Nash equilibria, and which can be implemented without division [41].

Of course FIXP contains PPAD, since it contains its complete problems, for example two-player Nash. Actually, the piecewise linear fragment of FIXP corresponds exactly to PPAD. Let *Linear-FIXP* be the class of problems that can be expressed as exact fixed point problems for functions given by algebraic circuits using $\{+, -, \max, \min\}$ (equivalently, $\{+, \max\}$) and multiplication with rational constants only; no division or multiplications of two gates or inputs is allowed. Then Linear-FIXP is equal to PPAD [41].

In several problems, we want a specific fixed point of a system $x = F(x)$, not just any fixed point. In particular, in several of the problems discussed in Section 4 for example, the function F is a monotone operator and we want a Least Fixed Point. To place such a problem in FIXP, one has to restrict the domain in a suitable, but polynomial-time computable, way so that only the desired fixed point is left in the domain. For some problems we know how to do this (for example, extinction probabilities of branching processes), but for others (e.g. recursive Markov chains) it is not clear that this can be done in polynomial time. In any case, the paradigm of the LFP of a monotone operator is one that appears in many common settings, and which deserves its own separate treatment.

7. Conclusions

Many problems, from a broad, diverse range of areas, involve the computation of an equilibrium or fixed point of some kind. There is a long line of research (both mathematical and algorithmic) in each of these areas, but for many of these basic problems we still do not have polynomial-time algorithms, nor do we have hard evidence of intractability (such as NP-hardness). We have reviewed a number of such problems here, and we discussed three complexity classes, PLS, PPAD and FIXP, that capture essential aspects of several types of such problems. The classes PLS and PPAD lie somewhere between P and TFNP (total search problems in NP), and FIXP (more precisely, the associated discrete problems) lie between P and PSPACE. These, and the obvious containment PPAD $\subseteq$ FIXP, are the only relationships we currently know between these classes and the other standard complexity classes. These relations are illustrated in Fig. 2. It would be very interesting and important to improve on this state of knowledge. Furthermore, there are several important problems that are in these classes, but are not (known to be) complete, so it is possible that one can make progress on them, without resolving the relation of the classes themselves.

REFERENCES

[1] L. Blum, F. Cucker, M. Shub, S. Smale, Complexity and Real Computation, Springer-Verlag, 1998.
[2] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc. Natl. Acad. Sci. 79 (1982) 2554–2558.
[3] D.S. Johnson, C.H. Papadimitriou, M. Yannakakis, How easy is local search? J. Comput. System Sci. 37 (1988) 79–100.
[4] M. Yannakakis, Computational complexity of local search, in: E.H.L. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, John Wiley, 1997.
[5] A. Schäffer, M. Yannakakis, Simple local search problems that are hard to solve, SIAM J. Comput. 20 (1991) 56–87.
[6] J. Nash, Non-cooperative games, Ann. of Math. 54 (1951) 289–295.
[7] M. Kearns, Graphical games, in: [72], 2007, pp. 159–180.
[8] R.W. Rosenthal, A class of games possessing pure-strategy Nash equilibria, Internat. J. Game Theory 2 (1973) 65–67.
[9] A. Fabrikant, C.H. Papadimitriou, K. Talwar, The complexity of pure Nash equilibria, in: Proc. ACM STOC, 2004, pp. 604–612.
[10] H. Ackermann, H. Roglin, B. Vöcking, On the impact of combinatorial structure on congestion games, in: Proc. 47th IEEE FOCS, 2006.
[11] A. Skopalik, B. Vöcking, Inapproximability of pure Nash equilibria, in: Proc. 40th ACM STOC, 2008, pp. 355–364.
[12] B. Vöcking, Congestion games: Optimization in competition, in: Proc. 2nd ACiD, 2006, pp. 9–20.
[13] A. Condon, The complexity of stochastic games, Inform. Comput. 96 (2) (1992) 203–224.
[14] A. Condon, On algorithms for simple stochastic games, Adv. Comp. Comp. Th., DIMACS 13 (1993) 51–73.
[15] M. Yannakakis, The analysis of local search problems and their heuristics, in: Proc. STACS, 1990, pp. 298–311.
[16] A. Ehrenfeucht, J. Mycielski, Positional strategies for mean payoff games, Internat. J. Game Theory 8 (1979) 109–113.
[17] U. Zwick, M.S. Paterson, The complexity of mean payoff games on graphs, Theoret. Comput. Sci. 158 (1996) 343–359.
[18] E.A. Emerson, C. Jutla, Tree automata, μ-calculus and determinacy, in: Proc. IEEE FOCS, 1991, pp. 368-377.
[19] A. Puri, Theory of hybrid systems and discrete event systems, Ph.D. Thesis, UC Berkeley, 1995.
[20] M. Jurdzinski, Deciding the winner in parity games is in UP∩coUP, Inform. Process. Lett. 68 (1998) 119–124.
[21] H. Scarf, The Computation of Economic Equilibria, Yale University Press, 1973.
[22] H. Uzawa, Walras' existence theorem and Brower's fixpoint theorem, Econom. Stud. Quart. 13 (1962) 59–62.
[23] K.J. Arrow, G. Debreu, Existence of an equilibrium for a competitive economy, Econometrica 22 (1954) 265–290.
[24] J. Geanakoplos, Nash and Walras equilibrium via Brouwer, Econom. Theory 21 (2003) 585–603.

[25] G. Debreu, Excess demand functions, J. Math. Econom. 1 (1974) 15–21.
[26] L.S. Shapley, Stochastic games, Proc. Natl. Acad. Sci. 39 (1953) 1095–1100.
[27] J. Filar, K. Vrieze, Competitive Markov Decision Processes, Springer, 1997.
[28] A. Neyman, S. Sorin (Eds.), Stochastic Games and Applications, Kluwer, 2003.
[29] T.E. Harris, The Theory of Branching Processes, Springer-Verlag, 1963.
[30] P. Haccou, P. Jagers, V.A. Vatutin, Branching Processes: Variation, Growth, and Extinction of Populations, Cambridge U. Press, 2005.
[31] M. Kimmel, D.E. Axelrod, Branching Processes in Biology, Springer, 2002.
[32] C. Manning, H. Schütze, Foundations of Statistical Natural Language Processing, MIT Press, 1999.
[33] G. Latouche, V. Ramaswami, Introduction to Matrix Analytic Methods in Stochastic Modeling, in: ASA-SIAM Series on Statistics and Applied Probability, 1999.
[34] K. Etessami, D. Wojtczak, M. Yannakakis, Quasi-birth-death processes, tree-like QBDs, probabilistic 1-counter automata, and pushdown systems, in: Proc. 5th Int. Symp on Quantitative Evaluation of Systems, 2008, pp. 243–253.
[35] R. Fagin, A. Karlin, J. Kleinberg, P. Raghavan, S. Rajagopalan, R. Rubinfeld, M. Sudan, A. Tomkins, Random walks with "back buttons", Ann. Appl. Probab. 11 (2001) 810–862.
[36] K. Etessami, M. Yannakakis, Recursive Markov chains, stochastic grammars, and monotone systems of non-linear equations, J. ACM 56 (1) (2009).
[37] J. Esparza, A. Kučera, R. Mayr, Model checking probabilistic pushdown automata, in: Proc. of 19th IEEE LICS'04, 2004.
[38] K. Etessami, M. Yannakakis, Efficient qualitative analysis of classes of recursive Markov decision processes and simple stochastic games., in: Proc. 23rd STACS, Springer, 2006.
[39] K. Etessami, M. Yannakakis, Recursive concurrent stochastic games, in: Proc. 33rd ICALP, 2006.
[40] K. Etessami, M. Yannakakis, Recursive Markov decision processes and recursive stochastic games, in: Proc. 32nd ICALP, 2005.
[41] K. Etessami, M. Yannakakis, On the complexity of Nash equilibria and other fixed points, in: Proc. IEEE FOCS, 2007.
[42] H. Scarf, The approximation of fixed points of a continuous mapping, SIAM J. Appl. Math. 15 (1967) 1328–1343.
[43] M.D. Hirsch, C.H. Papadimitriou, S.A. Vavasis, Exponential lower bounds for finding Brower fixed points, J. Complexity 5 (1989) 379–416.
[44] K. Sikorski, Optimal Solution of Nonlinear Equations, Oxford Univ. Press, 2001.
[45] K.-I. Ko, Computational complexity of fixpoints and intersection points, J. Complexity 11 (1995) 265–292.
[46] M. Richter, K.-C. Wong, Non-computability of competitive equilibrium, Econom. Theory 14 (1999) 1–27.
[47] O. Morgestern, J. von Neumann, The Theory of Games and Economic Behavior, Princeton Univ. Press, 1947.
[48] I. Gilboa, E. Zemel, Nash and correlated equilibria: Some complexity considerations, Games Econom. Behav. 1 (1989) 80–93.
[49] C. Lemke, J. Howson, Equilibrium points of bimatrix games, J. SIAM (1964) 413–423.
[50] R. Savani, B. von Stengel, Hard to solve bimatrix games, Econometrica 74 (2006) 397–429.
[51] C. Papadimitriou, On the complexity of the parity argument and other inefficient proofs of existence., J. Comput. System Sci. 48 (3) (1994) 498–532.
[52] C. Daskalakis, P. Goldberg, C. Papadimitriou, The complexity of computing a Nash equilibrium, in: Proc. ACM STOC, 2006, pp. 71–78.
[53] X. Chen, X. Deng, Settling the complexity of two-player Nash equilibrium, in: Proc. 47th IEEE FOCS, 2006, pp. 261–272.
[54] X. Chen, X. Deng, S.H. Teng, Computing Nash equilibria: Approximation and smoothed complexity, in: Proc. 47th IEEE FOCS, 2006, pp. 603–612.
[55] X. Chen, S.H. Teng, P. Valiant, The approximation complexity of win–lose games, in: Proc. 18th ACM SODA, 2007.
[56] R.J. Lipton, E. Markakis, A. Mehta, Playing large games using simple strategies, in: Proc. ACM Conf. Elec. Comm., 2003, pp. 36-41.
[57] P. Spirakis, Approximate equilibria for strategic two-person games, in: Proc. 1st Symp. Alg. Game Theory, 2008, pp. 5–21.
[58] X. Chen, X. Deng, On the complexity of 2d discrete fixed point problem, in: Proc. ICALP, 2006, pp. 489–599.
[59] B. Codenotti, A. Saberi, K. Varadarajan, Y. Ye, The complexity of equilibria: Hardness results for economies via a correspondence with games, Theoret. Comput. Sci. 408 (2008) 188–198.
[60] M.R. Garey, R.L. Graham, D.S. Johnson, Some NP-complete geometric problems, in: Proc. 8th ACM STOC, 1976, pp. 10–22.
[61] P. Tiwari, A problem that is easier to solve on the unit-cost algebraic RAM, J. Complexity (1992) 393–397.
[62] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, P.B. Miltersen, On the complexity of numerical analysis. in: Proc. 21st IEEE Comp. Compl. Conf., 2006.
[63] P. van Emde Boas, Machine models and simulations, in: J. van Leeuwen (Ed.), in: Handbook of Theoretical Computer Science, vol. A, MIT Press, 1990, pp. 1–66.
[64] J. Canny, Some algebraic and geometric computations in PSPACE, in: Proc. ACM STOC, 1988, pp. 460–467.
[65] J. Renegar, On the computational complexity and geometry of the first-order theory of the reals, parts I-III, J. Symbolic Comput. 13 (3) (1992) 255–352.
[66] R.J. Aumann, S. Hart (Eds.), Handbook of Game Theory, vol. 3, North-Holland, 2002.
[67] X. Chen, X. Deng, On algorithms for discrete and approximate Brower fixed points, in: Proc. ACM STOC, 2005, pp. 323–330.
[68] C. Daskalakis, A. Mehta, C. Papadimitriou, Progress in approximate Nash equilibria, in: Proc. 8th ACM Conf. Elec Comm., 2007, pp. 355–358.
[69] D.S. Johnson, The NP-completeness column: Finding needles in haystacks, ACM Trans. Algorithms 3 (2007).
[70] A.N. Kolmogorov, B.A. Sevastyanov, The calculation of final probabilities for branching random processes, Doklady 56 (1947) 783–786 (in Russian).
[71] S. Kontogiannis, P. Spirakis, Efficient algorithms for constant well-supported approximate equilibria in bimatrix games, in: Proc. 34th ICALP, 2007, pp. 286-296.
[72] N. Nisan, T. Roughgarden, E. Tardos, V. Vazirani, Algorithmic Game Theory, Cambridge Univ. Press, 2007.
[73] C. Papadimitriou, The Euclidean traveling salesman problem is NP-complete, Theoret. Comput. Sci. 4 (1977) 237–244.