# Fast Algorithms for Finding Randomized Strategies in Game Trees*

Daphne Koller† daphne@cs.berkeley.edu

Nimrod Megiddo‡ megiddo@almaden.ibm.com

Bernhard von Stengel§ i51bbvs@rz.unibw-muenchen.de

## Abstract

Interactions among agents can be conveniently described by game trees. In order to analyze a game, it is important to derive optimal (or equilibrium) strategies for the different players. The standard approach to finding such strategies in games with imperfect information is, in general, computationally intractable. The approach is to generate the *normal form* of the game (the matrix containing the payoff for each strategy combination), and then solve a linear program (LP) or a linear complementarity problem (LCP). The size of the normal form, however, is typically exponential in the size of the game tree, thus making this method impractical in all but the simplest cases. This paper describes a new representation of strategies which results in a practical linear formulation of the problem of two-player games with perfect recall (i.e., games where players never forget anything, which is a standard assumption). Standard LP or LCP solvers can then be applied to find optimal randomized strategies. The resulting algorithms are, in general, exponentially better than the standard ones, both in terms of time and in terms of space.

*In Proceedings of the 26th Annual ACM Symposium on Theory of Computing, 1994

†Computer Science Division, University of California, Berkeley, CA 94720; and IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120

‡IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120; and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel.

§Informatik 5, University of the Federal Armed Forces at Munich, 85577 Neubiberg, Germany.

Research supported in part by ONR Contracts N00014-91-C-0026 and N00014-94-C-0007, by the Air Force Office of Scientific Research (AFSC) under Contract F49620-91-C-0080, and by the Volkswagen Foundation. Some of the work was performed while the first author was at Stanford University. The United States Government is authorized to reproduce and distribute reprints for governmental purposes.

**In:** Proceedings of the 26th ACM Symposium on the Theory of Computing, 1994, 750–759

## 1 Introduction

Game theory models and analyzes situations involving conflict and cooperation among independent decision makers. It has played a substantial role in economics [1], and has been applied to biology [19], safeguards systems [2] and in the military [18], among other areas.

In computer science, the idea of modeling interactive situations as games is becoming more common. The classic paper of Chandra, Kozen and Stockmeyer [5] characterized the class PSPACE in terms of two-player games. The later work on interactive proof systems [34, 7] is also best understood in those terms. Reif [31] extends the paradigm of interactive proofs to a more general class of games. Worst-case analysis of algorithms can also be viewed naturally as a game between the solver and an adversary. For example, Yao's technique of proving lower bounds for randomized algorithms [39] follows directly from the classical minimax theorem [36] from game theory. A number of recent results in online computation are based on game-theoretic techniques (most obviously [3]). Game theory has been used in artificial intelligence to model interactions among intelligent agents, for example [32]. In the context of distributed systems, such issues as collective coin flipping [4] and privacy and security [14] have been analyzed using games. Other interactions relevant to computer science, such as network routing or load-sharing in distributed systems, also seem to fit naturally in a game-theoretic framework.

If an interactive situation is described as a game, a formal analysis should find optimal (or payoff maximizing) strategies for the players, and determine the expected outcome of the game. This analysis must take into account that the payoff for a player's strategy depends on the strategies of the other players. This is inherent in game-theoretic reasoning. A *solution* to a game is a certain combination of strategies, possibly randomized, one for each player. This combination is a *Nash equilibrium* if it is self-enforcing—no player can gain by unilaterally deviating from it (see Section 3). The Nash equilibrium is the central solution concept of noncooperative game theory. It reflects the modern paradigm that all players act individually, and cooperative acts (such as signing a contract) are explicitly modeled and must be rational for each player. As shown by Nash [30], every game has an equilibrium, possibly requiring the use of randomized strategies. The algorithmic problem is to find one.

In this paper, we investigate the problem of solving two-person games represented as game trees. We present algorithms both for zero-sum games and general games. (A game is zero-sum if the sum of the payoffs to the two players is always zero, so their interests are in conflict.) Our algorithms are, in general, exponentially better than the standard approach to the problem. They thus provide the first practical method for solving games that are not toy problems nor have a special structure.

The complexity of solving a game depends highly on the form in which the game is represented. A game may be described by its rules, by a tree or by payoff matrices. The conversion from a tree to a matrix may increase the size exponentially. The increase in size when a tree is generated from rules may not even be bounded. Thus it is not surprising that games are typically hard to solve when represented using their rules. For example, it is PSPACE-hard to find optimal strategies for generalized Go [27] and a variety of other games [33, 11], and EXPTIME-complete to find optimal strategies for generalized Chess [12] or generalized Checkers [13].

A game in *extensive form* is represented as a tree (a formal definition is given in Section 2). The nodes of the tree are game states (e.g., chess board positions together with the history of the game), the branchings result from the players' actions. The tree structure sometimes allows us to use *backward induction* as a solution method: Intuitively, backward induction computes optimal moves at each node of the tree, assuming that optimal moves at all of its descendants have already been computed. For games with *perfect information*, this idea is precisely the well-known max-min algorithm, due to Zermelo [40]. (A game has perfect information if at each point in the game all players know the entire history of the game up to that point. Thus, chess is a game with perfect information while poker is not.) In many situations to which game theory applies, the game tree is explicitly given; in such cases, it is typically not too large, so that the max-min algorithm is an effective computational approach. There has been a great deal of work on heuristics for reducing the running time of max-min search on larger game trees, where an exhaustive traversal of the entire tree is infeasible.

Game trees also allow us to represent situations where players have *imperfect information* about the present state of the game. In general, the solution to an imperfect-information game typically involves randomized strategies, a complication which is clearly unnecessary in the case of perfect information. Unfortunately, backward induction does not suffice for solving such games. The typical algorithms for solving extensive-form games with imperfect information operate by converting the game tree into normal form.

A game in *normal form* is represented by a matrix which describes the payoffs to the players under each possible strategy combination. A two-player zero-sum game in normal form can be solved in polynomial time *in the size of the matrix* using linear programming. In fact, solving such games is easily shown to be equivalent to general linear programming, so that the problem is $\mathcal{P}$-complete. A Nash equilibrium of a general two-player game in normal form can be found by solving a linear complementarity problem (LCP). The associated LCP can be solved by Lemke's [25] or the related Lemke-Howson algorithm [26], whose running time is in the worst case *exponential* in the size of the normal form, but which seems to work better in practice. The problem of finding an equilibrium is known neither to belong to $\mathcal{P}$ nor to be $\mathcal{NP}$-hard. The complexity of various associated problems has been studied in [16, 20, 15].

Applied to extensive-form games, these normal-form algorithms are typically impractical for any but toy problems. The reason is that the size of the normal form is, in general, exponential in the size of the game tree: each possible combination of moves determines a strategy and thus generates a row or column of the matrix. Even for game trees with a few dozen nodes, the resulting normal form is huge, so that it is infeasible to solve the resulting LP or LCP. This computational difficulty has often forced analysts to find solution techniques tailored to specific games [24] or even to abandon the game-theoretical approach altogether [28].

There has been some work on the problem of solving extensive form games directly. Thereby, backward induction is used to generate strategies for LP or LCP solvers dynamically from the game tree. Wilson [38] presented an algorithm that directly solves two-person games with *perfect recall*, where the players do not forget facts they once knew. While Wilson conjectured that his algorithm is efficient, he was not able to prove this fact (see Section 4). Koller and Megiddo [21] presented the first polynomial time algorithm for solving zero-sum games with perfect recall. Their algorithm, however, is based on the ellipsoid algorithm for linear programming [17], and is therefore not very practical. They also showed that, in games with *imperfect recall*, the problem of solving such games is $\mathcal{NP}$-hard.

In this paper, we present a method that avoids the exponential blowup of the normal-form transformation. The basic idea is that the outcome of the game depends only on the distribution of probability weights that a randomized strategy induces on the leaves of the tree. We represent a strategy compactly in terms of these *realization weights*, as introduced in [21]. These are defined directly in terms of the game tree, so their total size is linear rather than exponential in its size. As we show, this compact representation has a number of advantages. It can be used to construct a simple exponential-time algorithm for solving arbitrary two-person games (even with imperfect recall). It can also be used to provide a proof that Wilson's algorithm [38] is efficient. Also under the assumption of perfect recall, we can obtain further results. Then, the realization weights can be defined by a small system of linear constraints [21, 37]. In fact, the matrix representing this system is sparse and of *linear* size in the size of the game tree if stored sparsely. In [37], von Stengel noted that the same holds for the payoff matrices if the players are treated symmetrically. Using realization weights and LP duality, equilibrium strategies can then be found by solving a corresponding LP or LCP. We obtain the following two major results:

**Theorem 1.1** *The optimal strategies of a two-player zero-sum perfect-recall game in extensive form are the solutions of a linear program whose size, in sparse representation, is linear in the size of the game tree.*

**Theorem 1.2** *The Nash equilibria of a general two-player perfect-recall game in extensive form are the solutions of a linear complementarity problem whose size, in sparse representation, is linear in the size of the game tree. The problem of finding an equilibrium can be solved by Lemke's algorithm.*

We therefore obtain an efficient polynomial-time algorithm in the zero-sum case, and an often efficient exponential-time algorithm in the general case. In both cases, the complexity is measured in terms of the size of the game tree. Our algorithms are therefore exponentially better than the standard ones both in terms of time and in terms of space.

## 2 Extensive-Form Games

This section recalls the standard definition of a game in *extensive form* [23]. The basic structure of the game is a finite directed tree whose nodes denote game states. A play of the game starts at the root, proceeds according to the players' actions, and ends at a leaf. Extensive-form games can have any finite number of players, but we will concentrate on the two-player case. We will use the pronouns "she" for the first player and "he" for the second one. The internal nodes of the tree are of three types: decision nodes of player 1, decision nodes of player 2, and nodes for *chance moves*. The outgoing edges at a decision node represent possible actions at that node, and have distinct labels called *choices*. A *play* denotes the path from the root to some leaf. A *move* is a choice taken on that path. The *payoff* function $h$ determines a payoff vector $h(a) \in \mathbb{R}^2$ for each leaf $a$. The $k$th component $h^k(a)$ of $h(a)$ is the payoff at $a$ to player $k$. The relation between the payoffs to the two players is, in general, arbitrary. Thus, the interests of the players may coincide in some circumstances, and conflict in others. A *zero-sum* game models a situation where the interests of the players are strictly opposed in the sense that $h^2 = -h^1$; such a game is shown in Fig. 1. In the zero-sum case, the payoffs to the first player suffice to describe the game.

In order to represent situations where players may not know everything that occurs in the game (for example, in the game of poker one player's hand is not known to other players), the set of decision nodes is partitioned into *information sets*. Each information set $u$ belongs to exactly one player $k$. Intuitively, the player cannot differentiate between different nodes in the same information set. This implies that at each node $a$ in $u$, the player must have the same set $C_u$ of *choices* (labels for the outgoing edges) at $u$. For simplicity, it is assumed that the choice sets $C_u$ and $C_{u'}$ of any two information sets $u$ and $u'$ are disjoint. The set of all information sets of player $k$ is denoted by $U^k$. The set $\bigcup_{u \in U^k} C_u$ of all choices of player $k$ is denoted by $D^k$. We make the standard assumption that an information set is not visited more than once during one play. Finally, the behavior at the chance nodes is specified by a function $\beta$ which defines a probability distribution over the possible choices at each chance node.

The players plan their actions according to *strategies*. The basic strategy in an extensive-form game is a *pure* (or deterministic) strategy. A pure strategy $\pi^k$ of player $k$ specifies a choice at each information set $u \in U^k$. (Note that since the player knows only which information set he or she is at, a strategy must dictate the same move at all nodes in an information set.) Let $P^k$ denote the set of player $k$'s pure strategies. Clearly, this set is the cartesian product $\prod_{u \in U^k} C_u$. A *mixed strategy* $\mu^k$ of player $k$ is a probability distribution on the set $P^k$ of his or her pure strategies. We can define the expected payoff $H(\mu)$ for each pair of mixed strategies $\mu = (\mu^1, \mu^2)$. A strategy pair $\mu$, together with $\beta$, induces a probability distribution on the leaves of the tree. We denote the probability for reaching a leaf $a$ by $Pr_\mu(a)$. The expected payoff $H(\mu)$ is then defined to be

$$H(\mu) = \sum_a Pr_\mu(a) h(a). \qquad (1)$$

## 3 Solving Games

A *solution* to a game is a pair of strategies, one for each player. It should satisfy the minimal requirement of

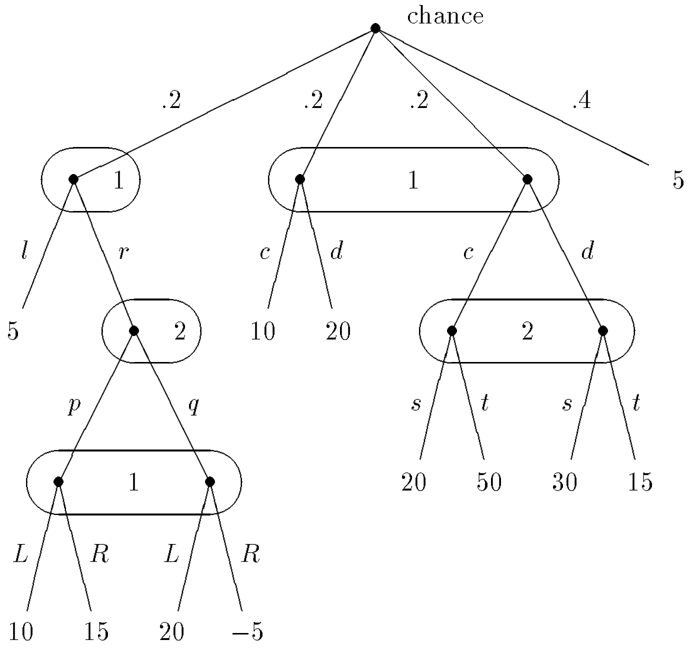


**Figure 1.** A zero-sum game in extensive form. From the root of the game tree, there is a chance move with the indicated probabilities. The ovals denote information sets, with the player to move written as a number inside. The leaves show the payoffs to player 1.

| | $(p,s)$ | $(p,t)$ | $(q,s)$ | $(q,t)$ |
|---|---|---|---|---|
| $(l,L,c)$ | 9 | 15 | 9 | 15 |
| $(l,L,d)$ | 13 | 10 | 13 | 10 |
| $(l,R,c)$ | 9 | 15 | 9 | 15 |
| $(l,R,d)$ | 13 | 10 | 13 | 10 |
| $(r,L,c)$ | 8 | 14 | 10 | 16 |
| $(r,L,d)$ | 10 | 7 | 12 | 9 |
| $(r,R,c)$ | 9 | 15 | 5 | 11 |
| $(r,R,d)$ | 11 | 8 | 7 | 4 |

**Figure 2.** Normal form of the game in Fig. 1. The rows and columns denote the pure strategies of player 1 and 2. The matrix entries are the expected payoffs to player 1. One of the first two pairs of rows is redundant because pure strategies with the choices $l, L$ and $l, R$ have the same effect. They are identified in the reduced normal form.

*equilibrium*: having been given the solution, no player should be able to gain by unilaterally deviating from it. Technically, a strategy pair $(\mu^1, \mu^2)$ is an equilibrium if $\mu^1$ is a best response to $\mu^2$ and $\mu^2$ is a best response to $\mu^1$, where a strategy $\mu^1$ is a *best response* to the strategy $\mu^2$ of player 2 if it yields the maximum possible payoff $H^1(\mu^1, \mu^2)$ to player 1 against $\mu^2$ (a best response of player 2 is defined analogously). Nash [30] proved that every game has such an equilibrium.

For zero-sum games, the equilibrium is a particularly strong solution concept. There, it is equivalent to a pair of max-min strategies, where each player optimizes his or her worst-case expected payoff. A max-min strategy for player 1 is a strategy $\mu^1$ that maximizes $\min_{\mu^2} H^1(\mu^1, \mu^2)$. A max-min strategy for player 2 is defined similarly. Hence, in a zero-sum game, the best worst case payoff is also the best payoff that a player can expect against a rational opponent. Furthermore, a player can choose to play his or her max-min strategy independently of the actions of the other player. In general two-player games, equilibrium strategies do not have these properties.

The standard method for solving any extensive-form game calls for constructing the *normal form* (also known as *strategic form*). In this form, all possible pure strategy pairs of the players are tabulated, along with the expected payoff for each player when such a strategy pair is played. More precisely, assume that $|P^1| = n$ and $|P^2| = m$. In the case of general two-player games, also called bimatrix games, the normal form is a pair of $m \times n$ matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. A row represents a pure strategy $\pi^1 \in P^1$ of player 1, a column represents a simultaneously chosen pure strategy $\pi^2 \in P^2$ of player 2, and the corresponding entries in $\boldsymbol{A}$ and $\boldsymbol{B}$ are $H^1(\pi^1, \pi^2)$ and $H^2(\pi^1, \pi^2)$, respectively. In zero-sum games, $\boldsymbol{B} = -\boldsymbol{A}$, so the normal form of the game is completely specified by $\boldsymbol{A}$. Figure 2 shows such a matrix game. The normal form representation loses all of the structure encoded in a game tree. On the other hand, games in this form are quite simple to solve. The methods are well known. The following exposition, however, will allow us to apply the same principles to a much smaller strategic represention, described in Section 5 below.

Consider a mixed strategy $\mu^1$ and recall that, by definition, it is a probability distribution on $P^1$. We can represent $\mu^1$ as a vector $\boldsymbol{x}$ with $m$ components representing the probabilities $\mu^1(\pi^1)$ assigned by $\mu^1$ to the pure strategies $\pi^1$ in $P^1$. In fact, any nonnegative $m$-vector $\boldsymbol{x} = (x_1, \ldots, x_m)^T$ with $\sum_{i=1}^m x_i = 1$ describes a mixed strategy. Similarly, a mixed strategy $\mu^2$ can be represented by an $n$-vector $\boldsymbol{y}$. It is easy to see that the expected payoff $H^1(\mu^1, \mu^2)$ is equal to the matrix product $\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{y}$, and similarly $H^2(\mu^1, \mu^2) = \boldsymbol{x}^T \boldsymbol{B} \boldsymbol{y}$.

In order to derive algorithms for computing an equilibrium, we consider first the problem of finding a best-response strategy for one player against a *given* mixed strategy of the other player. For example, player 1 wishes to maximize her payoff against a given strategy $\boldsymbol{y}$ by choosing her decision variables $x_i$ suitably. We have seen that the vector $\boldsymbol{x}$ defines a mixed strategy $\mu^1$ iff it satisfies certain linear constraints. Besides the inequality $\boldsymbol{x} \geq \boldsymbol{0}$, this is just a single equation: $\boldsymbol{Ex} = \boldsymbol{e}$, where $\boldsymbol{E}$ is a $1 \times m$ matrix of 1's, and $\boldsymbol{e}$ is the scalar 1. We write this constraint in a general fashion, since later we will use a different representation to characterize mixed strategies: one with fewer decision variables $x_i$ but more than a single equation. The subsequent reasoning will be valid for an arbitrary matrix $\boldsymbol{E}$ and right-hand side $\boldsymbol{e}$. Similarly, $\boldsymbol{y}$ represents a mixed strategy $\mu^2$ of player 2 iff $\boldsymbol{y} \geq \boldsymbol{0}$ and $\boldsymbol{Fy} = \boldsymbol{f}$, where the $\boldsymbol{F}$ is a $1 \times n$ matrix of 1's and the right-hand side $\boldsymbol{f}$ is the scalar 1. Again, $\boldsymbol{F}$ and $\boldsymbol{f}$ will later be generalized.

We consider the problem of finding a best response $\boldsymbol{y}$ to player 1's strategy $\boldsymbol{x}$. This is a linear program:

$$\begin{array}{ll} \underset{\boldsymbol{y}}{\text{maximize}} & (\boldsymbol{x}^T\boldsymbol{B})\boldsymbol{y} \\ \text{subject to} & \boldsymbol{Fy} = \boldsymbol{f}\,, \\ & \boldsymbol{y} \geq \boldsymbol{0}\,. \end{array} \tag{2}$$

In the *dual* problem of (2), a new unconstrained variable is introduced for each equation, yielding a vector $\boldsymbol{q}$ with the same dimension as $\boldsymbol{F}$ and $\boldsymbol{f}$ (so $\boldsymbol{q}$ is in the present case just a scalar), and the roles of objective function and right-hand side are exchanged:

$$\begin{array}{ll} \underset{\boldsymbol{q}}{\text{minimize}} & \boldsymbol{q}^T\boldsymbol{f} \\ \text{subject to} & \boldsymbol{q}^T\boldsymbol{F} \geq \boldsymbol{x}^T\boldsymbol{B}\,. \end{array} \tag{3}$$

Any pair $\boldsymbol{y}, \boldsymbol{q}$ of feasible solutions of the primal (2) and its dual (3) obviously satisfies $\boldsymbol{q}^T\boldsymbol{f} = \boldsymbol{q}^T\boldsymbol{Fy} \geq (\boldsymbol{x}^T\boldsymbol{B})\boldsymbol{y}$. This is known as the weak duality theorem, stating that feasible primal and dual objective function values comprise mutual bounds. By the strong duality theorem [10], they are equal at optimality; that is, $\boldsymbol{y}, \boldsymbol{q}$ is an optimal pair iff

$$\boldsymbol{q}^T\boldsymbol{f} = \boldsymbol{q}^T\boldsymbol{Fy} = \boldsymbol{x}^T\boldsymbol{By}. \tag{4}$$

Analogously, a best response $\boldsymbol{x}$ to the strategy $\boldsymbol{y}$ of player 2 is a solution to the following problem:

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{maximize}} & \boldsymbol{x}^T(\boldsymbol{Ay}) \\ \text{subject to} & \boldsymbol{x}^T\boldsymbol{E}^T = \boldsymbol{e}^T\,, \\ & \boldsymbol{x} \quad \geq \boldsymbol{0}\,. \end{array} \tag{5}$$

The dual problem (6) uses the unconstrained vector $\boldsymbol{p}$, in the present case again just a scalar:

$$\begin{array}{ll} \underset{\boldsymbol{p}}{\text{minimize}} & \boldsymbol{e}^T\boldsymbol{p} \\ \text{subject to} & \boldsymbol{E}^T\boldsymbol{p} \geq \boldsymbol{Ay}\,. \end{array} \tag{6}$$

Again, a pair $\boldsymbol{x}, \boldsymbol{p}$ of feasible solutions to (5) and (6) satisfies $\boldsymbol{e}^T\boldsymbol{p} \geq \boldsymbol{x}^T\boldsymbol{Ay}$, and is optimal iff

$$\boldsymbol{e}^T\boldsymbol{p} = \boldsymbol{x}^T\boldsymbol{E}^T\boldsymbol{p} = \boldsymbol{x}^T\boldsymbol{Ay}\,. \tag{7}$$

Note that in both cases, the primal and dual programs have the same value. In the case of (5) and (6) this is the best payoff player 1 can achieve if player 2 plays $\boldsymbol{y}$.

In order to find an equilibrium, we need to find $\boldsymbol{x}$ and $\boldsymbol{y}$ such that each is a best response to the other. For a zero-sum game, this can be computed with the following LP which is essentially (6) but with variables $\boldsymbol{p}$ and $\boldsymbol{y}$:

$$\begin{array}{lrcl} \underset{\boldsymbol{y},\boldsymbol{p}}{\text{minimize}} & \boldsymbol{e}^T\boldsymbol{p} & & \\ \text{subject to} & -\boldsymbol{Ay} + \boldsymbol{E}^T\boldsymbol{p} & \geq & \boldsymbol{0} \\ & -\boldsymbol{Fy} & = & -\boldsymbol{f}\,, \\ & \boldsymbol{y} & \geq & \boldsymbol{0}\,. \end{array} \tag{8}$$

The intuition used in deriving this LP is as follows. Since the game is zero-sum, the optimal value $\boldsymbol{e}^T\boldsymbol{p}$ of (6) is the payoff that player 2, if he plays $\boldsymbol{y}$, has to give to player 1. The system (8) reflects player 2's wish to minimize this payoff. Hence, the result of (8) is a max-min strategy for him and thus part of an equilibrium because the game is zero-sum. The dual of (8) is

$$\begin{array}{lrcl} \underset{\boldsymbol{x},\boldsymbol{q}}{\text{maximize}} & -\boldsymbol{q}^T\boldsymbol{f} & & \\ \text{subject to} & \boldsymbol{x}^T(-\boldsymbol{A}) - \boldsymbol{q}^T\boldsymbol{F} & \leq & \boldsymbol{0} \\ & \boldsymbol{x}^T\boldsymbol{E}^T & = & \boldsymbol{e}^T\,, \\ & \boldsymbol{x} & \geq & \boldsymbol{0}\,. \end{array} \tag{9}$$

In a zero-sum game, $-\boldsymbol{A} = \boldsymbol{B}$, so (9) is just (3) but with variables $\boldsymbol{q}$ and $\boldsymbol{x}$. Optimal solutions $(\boldsymbol{y}, \boldsymbol{p})$ and $(\boldsymbol{x}, \boldsymbol{q})$ to (8) and (9) fulfill $\boldsymbol{e}^T\boldsymbol{p} = -\boldsymbol{q}^T\boldsymbol{f}$ by strong duality. Since $\boldsymbol{e}^T\boldsymbol{p} \geq \boldsymbol{x}^T\boldsymbol{Ay} = -\boldsymbol{x}^T\boldsymbol{By} \geq -\boldsymbol{q}^T\boldsymbol{f}$, (7) and (4) must hold, so that these solutions are also optimal for (2), (3), (5), and (6).

An equilibrium of the zero-sum game, that is, a pair of optimal solutions to (8) and (9), is simultaneously computed by the simplex algorithm [10]. The problem can of course be solved in polynomial time (in the size of the matrix) by any polynomial linear programming algorithm.

In the case of a non-zero-sum game, the problem of finding equilibrium strategies for the players defines the following problem: find $\boldsymbol{x} \geq \boldsymbol{0}$, $\boldsymbol{y} \geq \boldsymbol{0}$, $\boldsymbol{p}$, $\boldsymbol{q}$ so that

$$\begin{array}{rrrcl} & -\boldsymbol{Ay} + \boldsymbol{E}^T\boldsymbol{p} & & \geq & \boldsymbol{0} \\ -\boldsymbol{B}^T\boldsymbol{x} & & +\boldsymbol{F}^T\boldsymbol{q} & \geq & \boldsymbol{0} \\ -\boldsymbol{Ex} & & & = & -\boldsymbol{e} \\ & -\boldsymbol{Fy} & & = & -\boldsymbol{f}\,, \end{array} \tag{10}$$

and so that

$$\begin{aligned} \boldsymbol{x}^T(-\boldsymbol{Ay} + \boldsymbol{E}^T\boldsymbol{p}) &= 0, \\ \boldsymbol{y}^T(-\boldsymbol{B}^T\boldsymbol{x} + \boldsymbol{F}^T\boldsymbol{q}) &= 0\,. \end{aligned} \tag{11}$$

The orthogonality conditions (11) are equivalent to (7) and (4), and are known as complementary slackness conditions in linear programming.

The system (10) and (11) is a *linear complementarity problem* (LCP) [8]. All solutions to this system can be found with a simple complete enumeration scheme whose running time is exponential in the size of the matrix. Lemke's algorithm [25] finds one solution to this system, i.e., one equilibrium, without the complete enumeration; it is known that certain equilibria are "elusive" to this method. A closely related algorithm by Lemke and Howson [26] uses a slightly different LCP involving only $\boldsymbol{A}$ and $\boldsymbol{B}$; for a nice exposition see [35]. Termination of Lemke's algorithm for the LCP (10) is not proved in [25]. This is a special case of our Theorem 1.2 (see Section 5).

While Lemke's algorithm typically avoids complete enumeration, its worst-case running time is still exponential. Deciding if a general LCP has a solution is $\mathcal{NP}$-complete [6], but the present LCP always has one. As we mentioned in the introduction, it is not known whether an equilibrium of a bimatrix game can be found in polynomial time; the problem is also not known to be $\mathcal{NP}$-hard. The related $\mathcal{NP}$-hardness results of Gilboa and Zemel [16] concern questions that essentially require consideration of all equilibria. The complexity of finding a single equilibrium of such a game is an important open question that remains unresolved.

Unfortunately, for a game in extensive form, even if it is zero-sum, the algorithms described in this section are typically impractical. The reason is that the algorithms are based on the normal form of the game, which can be very large. By definition, the normal form lists all pure strategies of both players. This number is exponential, since each combination of choices defines a pure strategy. It is sometimes possible to reduce the size of the normal form by eliminating redundant rows or columns in the matrix [9]. These are caused by certain information sets that are unreachable due to an earlier choice of the player. Unfortunately, this *reduced normal form* may still have exponential size. This is also observed in practice [28, 38]. A further reduction of bimatrix games, based on elimination of payoff-dominated rows or columns has been studied in [20, 15].

## 4 Realization Weights

Mixed strategies assign a probability to each of the exponentially many pure strategies. It turns out that some of this information is redundant. Intuitively, the effect of a mixed strategy $\mu^k$ in a game is determined by its behavior at the leaves of the game tree. However, a strategy $\mu^k$ for a single player does not define a probability distribution on the leaves: the probability $Pr_\mu(a)$ also depends on the other player's strategy. We therefore need an alternative notion: *realization weights*.

Let $k$ be a player, $k = 1, 2$, and let $a$ be a node of the game tree. There is a unique path from the root to $a$. On this path, certain edges correspond to moves of player $k$. The string of labels of these edges is denoted by $\sigma^k(a)$ and is called the *sequence* of choices of player $k$ leading to $a$. It may be the empty sequence $\emptyset$, for example if $a$ is the root.

For the moment, we consider the leaves $a$. The sequence $\sigma^k(a)$ describes the choices that player $k$ has to make so that $a$ can be reached in the game. A pure strategy $\pi^k$ can only reach $a$ if it chooses to make every move in $\sigma^k(a)$ at the appropriate information sets. For a mixed strategy $\mu^k$, we define the *realization weight* of $a$ under $\mu^k$, denoted by $\mu^k(\sigma^k(a))$, to be the sum of the probabilities $\mu^k(\pi^k)$ over all those $\pi^k$ whose choices match those required by $\sigma^k(a)$. Let $\beta(a)$ denote the product of the chance probabilities on the path to $a$, and consider a pair $\mu = (\mu^1, \mu^2)$ of mixed strategies. When these strategies are used, then $a$ is obviously reached with probability

$$Pr_\mu(a) = \mu^1(\sigma^1(a))\, \mu^2(\sigma^2(a))\, \beta(a). \qquad (12)$$

Now, consider two mixed strategies $\mu^1$ and $\tilde{\mu}^1$ that generate the same realization weights. Then for any strategy $\mu^2$, the probabilities of reaching the leaves of the tree are the same. Therefore, by (1), the expected payoffs $H(\mu^1, \mu^2)$ and $H(\tilde{\mu}^1, \mu^2)$ are also equal. This justifies our intuition that the set $\{\mu^k(\sigma^k(a))\}_a$ of realization weights of the leaves $a$ captures all the relevant information about $\mu^k$. We call such strategies $\mu^1$ and $\tilde{\mu}^1$ *equivalent*. (This corresponds to a standard notion of equivalence of strategies [9].) A set of realization weights therefore describes an equivalence class of mixed strategies. Using techniques of [22], we can prove that this equivalence class contains a member that can be represented compactly. More precisely, define the *support* of $\mu^k$ to be the set of pure strategies to which it assigns positive probability.

**Theorem 4.1** *For any strategy $\mu^k$ there exists an equivalent strategy $\tilde{\mu}^k$ such that the support of $\tilde{\mu}^k$ contains at most $\ell$ pure strategies, where $\ell$ is the number of leaves in the tree.*

Therefore, although mixed strategies often use an exponential number of pure strategies, this is unnecessary. Mixed strategies need never be "too large": we need at most a linear number of pure strategies in the size of the game tree. There are several benefits to using a mixed strategy whose support is small: (i) we need less space to store it; (ii) it is easier to use while playing the game; (iii) algorithms that use the strategy (for example, in order to compute a best response to it) are more efficient;

(iv) it is easier to reason about and understand the behavior of the strategy in the game. But there is yet another important benefit we can derive from the existence of strategies with small supports. When computing a solution, certain algorithms traverse all possible supports for a potential solution strategy. In general, the number of possible supports is exponential in the size of normal form, and therefore doubly exponential in the size of the tree. However, we only need to look for strategies over *small* supports, and the number of small supports is exponential in the size of tree. For example, the complete enumeration algorithm for solving LCP's, mentioned in Section 3, can be transformed to run in exponential time *in the size of the game tree* rather than in the size of the normal form. This allows us to prove the following theorem:

**Theorem 4.2** *All the Nash equilibria of a general two-player game in extensive form can be found in exponential time in the size of the game tree.*

Currently, this is the only exponential-time algorithm for computing equilibria in general two-player games with *imperfect recall*. Player $k$ is said to have perfect recall if for any information set $u \in U^k$, any two nodes $a$ and $b$ in $u$ define the same sequence for player $k$, that is, $\sigma^k(a) = \sigma^k(b)$. Intuitively, if the player makes different choices on the paths to $a$ and $b$, and does not forget, then he or she must be able to distinguish between $a$ and $b$, so these nodes belong to different information sets.

As we mentioned in the introduction, Wilson [38] presented an algorithm for solving general two-player games with perfect recall. We can also use this idea to provide a formal analysis for this algorithm. Wilson's algorithm does not use the entire normal form. Instead, only pure strategies actually used in a mixed strategy are generated as needed from the game tree. (The technique for generating pure strategies from the game tree uses backward induction and hence requires the assumption of perfect recall.) Wilson justified this by the observation "commonly verified in computational experience on practical problems" that "the frequency of equilibria using only a very few of the available pure strategies is very high." Our results show that Wilson's assumption is provably true, rather than being an experimental observation. We can thus prove that Wilson's algorithm is indeed efficient (exponential in the size of the game tree).

## 5 Sequences Instead of Strategies

As we showed in the previous section, the concept of realization weights can be used to improve algorithms that use mixed strategies as well as the analysis of such algorithms. We can go further by computing directly with realization weights instead of mixed strategy probabilities. More precisely, in previous sections we considered algorithms that compute with vectors $\boldsymbol{x}, \boldsymbol{y}$ representing mixed strategies. We now present algorithms that compute with vectors that directly represent realization weights.

For our construction, we consider realization weights for sequences in general rather than for sequences associated with nodes. Let $S^k$ be the set of sequences for player $k$. We introduce a variable $x_\sigma$ for each $\sigma \in S^1$ and a variable $y_\sigma$ for each $\sigma \in S^2$, and denote the vectors with these components by $\boldsymbol{x}$ and $\boldsymbol{y}$. Intuitively, a component of $\boldsymbol{x}$ represents $\mu^1(\sigma)$ for some $\sigma \in S^1$ in the same way as it represented $\mu^1(\pi^1)$ for some $\pi^1 \in P^1$ in Section 3. The major problem in this context is in deciding whether a particular vector $\boldsymbol{x} = (x_\sigma)$ represents *some* mixed strategy $\mu^1$. In general, this problem is $\mathcal{NP}$-hard [22]. However, if the players have perfect recall, we can characterize realization weights by a small set of linear constraints [21]. As in Section 3, these constraints can be used to find equilibria as solutions to an LP or LCP, but of *small* size. Essentially, this corresponds to a strategic description of the game (called sequence form in [37]) where sequences replace pure strategies.[1]

For the remainder of this section, assume that both players have perfect recall. In this case, every sequence is either the empty sequence $\emptyset$ or the concatenation $\sigma_u c$ of a sequence $\sigma_u = \sigma^k(a)$ for some node $a \in u$ (for some $u \in U^k$) and of $c \in C_u$. (The sequence $\sigma_u$ is unique by perfect recall, and needs no superscript $k$ since we know $u \in U^k$.) In the latter case, the sequence is uniquely determined by the last choice $c$. The number of sequences in $S^k$ is therefore $1 + |D^k|$. Given this description of $S^k$, the following lemma and its analogue for player 2 are easy to prove:

**Lemma 5.1** *The weights $x_\sigma$ for $\sigma \in S^1$ are induced by a mixed strategy $\mu^1$ iff they are nonnegative, $x_\emptyset = 1$, and $x_{\sigma_u} = \sum_{c \in C_u} x_{\sigma_u c}$ for all information sets $u \in U^1$.*

This lemma allows us to use the LP approach of Section 3. The vector $\boldsymbol{x}$ represents a mixed strategy iff it satisfies the $1 + |U^1|$ linear equations described in Lemma 5.1. Mixed strategies of player 1 can thus be represented by a small number, namely $1+|D^1|$, of variables, characterized by a small number, namely $1+|U^1|$,

[1] For games with perfect recall, realization weights are closely related to *behavior strategies*. These use local randomization, by describing a probability distribution over the choices at each information set. In games with perfect recall, for every mixed strategy there is an equivalent behavior strategy [23], so that we could use behavior strategies as a compact representation of a mixed strategy. However, behavior strategies are not suitable for the purposes of computing equilibria, since the payoff cannot be described as a linear function in the variables defining a behavior strategy.

of linear constraints. We can therefore define a constraint matrix $\boldsymbol{E}$ and a right-hand-side $\boldsymbol{e}$ such that the weights in $\boldsymbol{x}$ represent some mixed strategy iff $\boldsymbol{E}\boldsymbol{x} = \boldsymbol{e}$ and $\boldsymbol{x} \geq \mathbf{0}$ hold. The dimensions of $\boldsymbol{E}$ are linear in the size of the tree because $\boldsymbol{E}$ is of size $(1+|U^1|)\times(1+|D^1|)$. Furthermore, the matrix is sparse since there is one entry 1 for each column and one entry $-1$ for each row except the first, and all other entries are zero. A similar construction of $\boldsymbol{y}$, $\boldsymbol{F}$, and $\boldsymbol{f}$ goes through for player 2. Such a construction for both players is illustrated in Fig. 3.

The $(1+|D^1|)\times(1+|D^2|)$ payoff matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ for the two players are also easy to define. The payoff contribution of a pair of sequences $\sigma^1$ and $\sigma^2$ is determined by those leaves $a$ such that $\sigma^1(a) = \sigma^1$ and $\sigma^2(a) = \sigma^2$. More than one leaf $a$ may define the same pair of sequences due to chance moves. The entries of $\boldsymbol{A}$ and $\boldsymbol{B}$ are thus the sum over all such leaves $a$ of $\beta(a)h^1(a)$ and $\beta(a)h^2(a)$, respectively; as above, $\beta(a)$ is the product of chance probabilities along the path to $a$. Entries of $\boldsymbol{A}$ and $\boldsymbol{B}$ for which the sequences do not correspond to leaves are zero. Then, by (1) and (12), players playing according to $\boldsymbol{x}$ and $\boldsymbol{y}$ receive the expected payoffs $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{y}$ and $\boldsymbol{x}^T\boldsymbol{B}\boldsymbol{y}$. An example for $\boldsymbol{A}$ (where $\boldsymbol{B} = -\boldsymbol{A}$) is shown in Fig. 3. Note that these matrices have much smaller dimension and are easier to construct than the normal form matrices. They are also sparse, since there are at most linearly (as opposed to quadratically) many nonzero entries, at most one per leaf. In contrast, normal form matrices are full (compare Fig. 2). When solving a large LP or LCP, sparsity can be exploited successfully.

The argument of Section 3 can now be carried through without change. As a result, a zero-sum game in extensive form with perfect recall can be solved by an LP, namely (8), of small size, which shows Theorem 1.1. If the game has general payoffs, its equilibria are the solutions to the LCP (10). This shows Theorem 1.2, except for the claim that this LCP can be solved by Lemke's algorithm, which we will show in the remainder of this paper.

Lemke's algorithm [25] is a pivoting scheme similar to the simplex algorithm for linear programming, but with a different "complementarity" rule for the entering and leaving variables, and no objective function; for expositions see [29, 8]. Under certain conditions, the algorithm finds a solution $\boldsymbol{z} \in \mathbb{R}^N$ to the LCP specified by an $N \times N$ matrix $\boldsymbol{M}$ and a vector $\boldsymbol{b} \in \mathbb{R}^N$:

$$\begin{aligned} \boldsymbol{z} &\geq \mathbf{0} \\ \boldsymbol{b} + \boldsymbol{M}\boldsymbol{z} &\geq \mathbf{0} \\ \boldsymbol{z}^T(\boldsymbol{b} + \boldsymbol{M}\boldsymbol{z}) &= 0\,. \end{aligned} \tag{13}$$

The LCP (10), (11) can be brought to this standard form with $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{p}', \boldsymbol{p}'', \boldsymbol{q}', \boldsymbol{q}'')^T$, where $\boldsymbol{p}$ and $\boldsymbol{q}$ are represented as differences $\boldsymbol{p}' - \boldsymbol{p}''$ and $\boldsymbol{q}' - \boldsymbol{q}''$ of nonnegative vectors of the same dimension, respectively. Correspondingly, each of the equations in (10) is replaced by two inequalities, so that the constraints (10) have the form $\boldsymbol{b} + \boldsymbol{M}\boldsymbol{z} \geq \mathbf{0}$ with $\boldsymbol{b} = (\mathbf{0}, \mathbf{0}, \boldsymbol{e}, -\boldsymbol{e}, \boldsymbol{f}, -\boldsymbol{f})^T$.

It may happen that Lemke's algorithm does not terminate with a solution to the LCP (13). (The analogous phenomenon occurs with the simplex algorithm in case of an unbounded objective function.) One condition that prevents this possibility is asserted in the following Theorem 4.4.13 of [8].

**Theorem 5.2** *If (i) $\boldsymbol{z}^T\boldsymbol{M}\boldsymbol{z} \geq 0$ for all $\boldsymbol{z} \geq \mathbf{0}$, (ii) $\boldsymbol{z} \geq 0$, $\boldsymbol{M}\boldsymbol{z} \geq \mathbf{0}$ and $\boldsymbol{z}^T\boldsymbol{M}\boldsymbol{z} = 0$ imply $\boldsymbol{b}^T\boldsymbol{z} \geq 0$, and (iii) the problem is nondegenerate, then Lemke's algorithm computes a solution of the LCP (13).*

Nondegeneracy means that for each row $i = 1, \ldots, N$, either $\boldsymbol{z}_i$ or $(\boldsymbol{b} + \boldsymbol{M}\boldsymbol{z})_i$ is nonzero for any solution $\boldsymbol{z}$ to (13) and to a certain generalized system used in the algorithm. This can always be achieved by slightly perturbing $\boldsymbol{b}$, replacing it by $\boldsymbol{b} + (\varepsilon, \varepsilon^2, \ldots, \varepsilon^N)^T$ for any sufficiently small positive $\varepsilon$. Thereby, the solution is not changed by performing the pivoting operations as if $\varepsilon$ is "just vanishing", which is implemented by certain lexicographic comparisons only involving $\boldsymbol{b}$ and $\boldsymbol{M}$ (and no $\varepsilon$); see [8, p. 340]. A similar technique for prevention of cycling given a degenerate problem is known for the simplex algorithm. With degeneracy thus taken care of, Theorem 5.2 is employed using the following two lemmas, where the first is immediate from the structure of the constraint matrices (see Fig. 3 for an example).

**Lemma 5.3** *The only nonnegative solutions $\boldsymbol{x}$ and $\boldsymbol{y}$ to $\boldsymbol{E}\boldsymbol{x} = \mathbf{0}$ and $\boldsymbol{F}\boldsymbol{y} = \mathbf{0}$ are $\boldsymbol{x} = \mathbf{0}$ and $\boldsymbol{y} = \mathbf{0}$.*

**Lemma 5.4** *Let $\boldsymbol{p}$ and $\boldsymbol{q}$ be unconstrained. Then $\boldsymbol{E}^T\boldsymbol{p} \geq \mathbf{0}$ and $\boldsymbol{F}^T\boldsymbol{q} \geq \mathbf{0}$ imply $\boldsymbol{e}^T\boldsymbol{p} \geq 0$ and $\boldsymbol{f}^T\boldsymbol{q} \geq 0$.*

*Proof.* Consider the following LP: maximize 0 subject to $\boldsymbol{E}\boldsymbol{x} = \boldsymbol{e}$, $\boldsymbol{x} \geq \mathbf{0}$. It is feasible, so the value 0 of its objective function is a lower bound for the objective function of the dual LP: minimize $\boldsymbol{e}^T\boldsymbol{p}$ subject to $\boldsymbol{E}^T\boldsymbol{p} \geq \mathbf{0}$; hence $\boldsymbol{e}^T\boldsymbol{p} \geq 0$. Similarly, $\boldsymbol{F}^T\boldsymbol{q} \geq \mathbf{0}$ implies $\boldsymbol{f}^T\boldsymbol{q} \geq 0$. □

To show that the assumptions of Theorem 5.2 are met, let $\boldsymbol{z} \geq \mathbf{0}$. By (10), $\boldsymbol{z}^T\boldsymbol{M}\boldsymbol{z} = \boldsymbol{x}^T(-\boldsymbol{A} - \boldsymbol{B})\boldsymbol{y}$, which is nonnegative if $\boldsymbol{A}$ and $\boldsymbol{B}$ have no positive entries. This can be assumed without loss of generality, by subtracting a constant from the payoffs to the players at the leaves of the tree so that these become nonpositive (this transformation does not change the game). Furthermore, $\boldsymbol{M}\boldsymbol{z} \geq \mathbf{0}$ is equivalent to $-\boldsymbol{A}\boldsymbol{y} + \boldsymbol{E}^T\boldsymbol{p} \geq \mathbf{0}$, $-\boldsymbol{B}^T\boldsymbol{x} + \boldsymbol{F}^T\boldsymbol{q} \geq \mathbf{0}$, $\boldsymbol{E}\boldsymbol{x} = \mathbf{0}$ and $\boldsymbol{F}\boldsymbol{y} = \mathbf{0}$. Thus, by Lemma 5.3, $\boldsymbol{x} = \mathbf{0}$ and $\boldsymbol{y} = \mathbf{0}$, and

$$
\begin{array}{cc}
 & \begin{array}{ccccccc} \emptyset & l & r & rL & rR & c & d \end{array} \\
\boldsymbol{E} = & \left[\begin{array}{ccccccc} 1 & & & & & & \\ -1 & 1 & 1 & & & & \\ & & -1 & 1 & 1 & & \\ -1 & & & & & 1 & 1 \end{array}\right]
\end{array}
\qquad
\boldsymbol{e} = \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \end{array}\right]
$$

$$
\begin{array}{cc}
 & \begin{array}{ccccc} \emptyset & p & q & s & t \end{array} \\
\boldsymbol{F} = & \left[\begin{array}{ccccc} 1 & & & & \\ -1 & 1 & 1 & & \\ -1 & & & 1 & 1 \end{array}\right]
\end{array}
\qquad
\boldsymbol{f} = \left[\begin{array}{c} 1 \\ 0 \\ 0 \end{array}\right]
$$

$$
\begin{array}{ccc}
 & \begin{array}{ccccc} \emptyset & p & q & s & t \end{array} & \\
\boldsymbol{A} = & \left[\begin{array}{ccccc} 2 & & & & \\ 1 & & & & \\ & & & & \\ & 2 & 4 & & \\ & 3 & -1 & & \\ 2 & & & 4 & 10 \\ 4 & & & 6 & 3 \end{array}\right] & \begin{array}{c} \emptyset \\ l \\ r \\ rL \\ rR \\ c \\ d \end{array}
\end{array}
$$

**Figure 3.** Constraint matrices and payoff matrix for the realization weights of the sequences in the game in Fig. 1. The matrices are sparse, zero entries are omitted.

therefore $\boldsymbol{E}^T\boldsymbol{p} \geq \boldsymbol{0}$ and $\boldsymbol{F}^T\boldsymbol{q} \geq \boldsymbol{0}$, and by Lemma 5.4, $\boldsymbol{b}^T\boldsymbol{z} = \boldsymbol{e}^T\boldsymbol{p} + \boldsymbol{f}^T\boldsymbol{q} \geq 0$. This shows that an equilibrium of the game is found by Lemke's algorithm.

## References


[1] R. J. Aumann and eds. S. Hart. *Handbook of Game Theory, Vol. 1.* North-Holland, Amsterdam, 1992.

[2] R. Avenhaus. *Safeguards Systems Analysis.* Plenum Press, New York, 1986.

[3] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. In *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, pages 379–386, 1990.

[4] M. Ben-Or and N. Linial. Collective coinf flipping, robust voting games and minima of Banzhaf values. In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 408–416, 1985.

[5] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[6] S. J. Chung. NP-completeness of the linear complementarity problem. *Journal of Optimization Theory and Applications*, 60:393–399, 1989.

[7] A. Condon, J. Feigenbaum, C. Lund, and P. Shor. Probabilistically checkable debate systems and approximation algorithms for PSPACE-hard functions (extended abstract). In *Proceedings of the 25nd Annual ACM Symposium on the Theory of Computing*, pages 305–314, 1993.

[8] R. W. Cottle, J.-S. Pang, and R. E. Stone. *The Linear Complementarity Problem.* Academic Press, 1992.

[9] N. Dalkey. Equivalence of information patterns and essentially indeterminate games. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games II*, pages 217–243. Princeton University Press, 1953.

[10] G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1963.

[11] S. Even and R. E. Tarjan. A combinatorial problem which is complete in polynomial space. *Journal of the Association for Computing Machinery*, 23:710–719, 1976.

[12] A. S. Fraenkel and D. Lichtenstein. Computing a perfect strategy for $n \times n$ Chess requires time exponential in $n$. *Journal of Combinatorial Theory, Series A*, 31:199–214, 1981.

[13] A. S. Fraenkel and D. Lichtenstein. $n$ by $n$ Checkers in EXPTIME complete. *SIAM Journal on Computing*, 13(2):252–267, 1984.

[14] M. Franklin, Z. Galil, and M. Yung. Eavesdropping games: A graph-theoretic approach to privacy in distributed systems. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 670–679, 1993.

[15] I. Gilboa, E. Kalai, and E. Zemel. The complexity of eliminating dominated strategies. *Mathematics of Operations Research*, 18(3):553–565, 1993.

[16] I. Gilboa and E. Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1:80–93, 1989.

[17] M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.

[18] O. Hájek. *Pursuit Games*. Academic Press, New York, 1975.

[19] J. Hofbauer and K. Sigmund. *The Theory of Evolution and Dynamical Systems*. Cambridge University Press, 1988.

[20] D. E. Knuth, C. H. Papadimitriou, and J. N. Tsitsiklis. A note on strategy elimination in bimatrix games. *Operations Research Letters*, 7:103–107, 1988.

[21] D. Koller and N. Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4:528–552, 1992.

[22] D. Koller and N. Megiddo. Finding small sample spaces satisfying given constraints. *SIAM Journal on Discrete Mathematics*, 1993. To appear.

[23] H. W. Kuhn. Extensive games and the problem of information. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games II*, pages 193–216. Princeton University Press, 1953.

[24] Z. F. Lansdowne, G. B. Dantzig, R. P. Harvey, and R. D. McKnight. Development of an algorithm to solve multi-stage games. Technical report, Control Analysis Corporation, Palo Alto, CA, 1973.

[25] C. E. Lemke. Bimatrix equilibrium points and mathematical programming. *Management Science*, 11:681–689, 1965.

[26] C. E. Lemke and Jr. J. T. Howson. Equilibrium points in bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12:413–423, 1964.

[27] D. Lichtenstein and M. Sipser. Go is polynomial-space hard. *Journal of the ACM*, 27:393–401, 1980.

[28] W. F. Lucas. An overview of the mathematical theory of games. *Management Science*, 15, Appendix P:3–19, 1972.

[29] K. G. Murty. *Linear Complementarity, Linear and Nonlinear Programming*. Heldermann Verlag, Berlin, 1988.

[30] J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.

[31] J. H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and Systems Sciences*, 29:274–301, 1984.

[32] J. S. Rosenschein. Consenting agents: Negotiation mechanisms for multi-agent systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 792–799, 1993.

[33] T. J. Schaefer. On the complexity of some two-person perfect-information games. *Journal of Computer and Systems Sciences*, 16:185–225, 1978.

[34] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39:869–877, 1992.

[35] L. S. Shapley. A note on the Lemke-Howson algorithm. *Mathematical Programming Study*, 1:175–189, 1974.

[36] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, 2nd edition, 1947.

[37] B. von Stengel. Lp representation and efficient computation of behavior strategies. Technical Report S-9301, University of the Federal Armed Forces at Munich, 1993.

[38] R. Wilson. Computing equilibria of two-person games from the extensive form. *Management Science*, 18:448–460, 1972.

[39] A. C. Yao. Probabilistic computation: Towards a unified measure of complexity. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1977.

[40] E. Zermelo. Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. In E. W. Hobson and A. E. H. Love, editors, *Proceedings of the Fifth International Congress of Mathematicians II*, pages 501–504. Cambridge University Press, 1913.