Computing Game-Theoretic Solutions and Applications to Security*

Vincent Conitzer
Departments of Computer Science and Economics
Duke University
Durham, NC 27708, USA
conitzer@cs.duke.edu

Abstract

The multiagent systems community has adopted game theory as a framework for the design of systems of multiple self-interested agents. For this to be effective, efficient algorithms must be designed to compute the solutions that game theory prescribes. In this paper, I summarize some of the state of the art on this topic, focusing particularly on how this line of work has contributed to several highly visible deployed security applications, developed at the University of Southern California.

Introduction

How should an agent choose its actions optimally? The standard decision-theoretic framework is to calculate a utility for every possible outcome, as well as, for each possible course of action, a conditional probability distribution over the possible outcomes. Then, the agent should choose an action that maximizes expected utility. However, this framework is not straightforward to apply when there are other rational agents in the environment, who have potentially different objectives. In this case, the agent needs to determine a probability distribution over the other agents' actions. To do so, it is natural to assume that those agents are themselves trying to optimize their actions for their own utility functions. But this quickly leads one into circularities: if the optimal action for agent 1 depends on agent 2's action, but the optimal action for agent 2 depends on agent 1's action, how can we ever come to a conclusion? This is precisely the type of problem that *game theory* aims to address. It allows agents to form beliefs over each other's actions and to act rationally on those beliefs in a consistent manner. The word "game" in game theory is used to refer to any strategic situation, including games in the common sense of the word, such as board and card games, but also important domains that are not so playful, such as the security games that we will discuss towards the end of this paper.

Social scientists and even biologists use game theory to model behaviors observed in the world. In contrast, AI researchers tend to be interested in constructively using game theory in the design of their agents, enabling them to reason strategically. For this to work, it is essential to design efficient *algorithms* for computing game-theoretic solutions—algorithms that take a game as input, and produce a way to play as output.

The rest of this paper is organized as follows. We first discuss various ways to represent games. We then discuss various solution concepts—notions of what it means to solve the game—and whether and how these solutions can be computed efficiently, with a focus on two-player normal-form games. Finally, we discuss exciting recent developments in the deployment of these techniques in high-value security applications.

Representing Games

We first discuss how to *represent* games. This is important first of all from a conceptual viewpoint: we cannot have general tools for analyzing games precisely without making the games themselves precise. There are several standard representation schemes from the game theory literature that allow us to think clearly about games. Additionally, the representation scheme of course has a major impact on scalability. From this perspective, a good representation scheme allows us to compactly represent natural games, in the same sense that Bayesian networks allow us to compactly represent many natural probability distributions; and it allows for efficient algorithms. While the standard representation schemes from game theory are sufficient in some cases, in other cases the representation size blows up exponentially.

In this section, we review standard representation schemes from game theory, as well as, briefly, representation schemes that have been introduced in the AI community to represent families of natural games more compactly. Discussion of compact representation schemes for security games is postponed towards the end of the paper.

Normal-form games

Perhaps the best-known representation scheme for games is the *normal form* (also known as the *strategic form*, or, in the case of two players, the *bimatrix form*). Here, each player i has a set of available *pure strategies* S_i, of which she must choose one. Each player i also has a *utility function* $u_i : S_1 \times \ldots \times S_n \rightarrow \mathbb{R}$ which maps outcomes (consisting of a

*This paper was invited as a "What's Hot" paper to the AAAI'12 Sub-Area Spotlights track.

Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

pure strategy for each player) to utilities. Two-player games are often written down as follows:

	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

This is the familiar game of rock-paper-scissors. Player 1 is the "row" player and selects a row, and player two is the "column" player and selects a column. The numbers in each entry are the utilities of the row and column players, respectively, for the corresponding outcome. Rock-paper-scissors is an example of a *zero-sum* game: in each entry, the numbers sum to zero, so whatever one player gains, the other loses.

Extensive-form games

Extensive-form games allow us to directly model the temporal and informational structure of a game. They are a generalization of the game trees familiar to most AI researchers. They allow chance nodes (often called "moves by Nature" in game theory), non-zero-sum outcomes, as well as *imperfect information*. In a perfect-information game, such as chess, checkers, or tic-tac-toe, nothing about the current state of the game is ever hidden from a player. In contrast, in (for example) most card games, no single player has full knowledge of the state of the game, because she does not know the cards of the other players. This is represented using *information sets*: an information set is a set of nodes in the game tree that all have the same acting player, such that the player cannot distinguish these nodes from each other.

For example, consider the following simple (perhaps the simplest possible interesting) poker game. We have a two(!)-card deck, with a King and a Jack. The former is a winning card, the latter a losing card. Both players put $1 into the pot at the outset. Player 1 draws a card; player 2 does not get a card. Player 1 must then choose to bet (put an additional $1 into the pot) or check (do nothing). Player 2 must then choose to call (match whatever additional amount player 2 put into the pot, possibly $0), or fold (in which case player 1 wins the pot). If player 2 calls, player 1 must show her card; if it is a King, she wins the pot, and if it is a Jack, player 2 wins the pot.

Figure 1 shows the extensive form of this game. The dashed lines between nodes for player 2 connect nodes inside the same information set, i.e., between which player 2 cannot tell the difference (because he does not know which card player 1 has).

It is possible to convert extensive-form games to normal-form games, as follows. A pure strategy in an extensive-form game consists of a plan that tells the player which action to take at *every* information set that she has in the game. (A player must choose the same action at every node inside the same information set, because after all she cannot distinguish them.) One pure strategy for player 1 is "Bet on a King, Check on a Jack," which we represent as BC. Similarly, one pure strategy for player 2 is "Fold on a Bet, Call on a Check," which we represent as FC. Given both players' pure strategies, we can calculate the expected utility for each

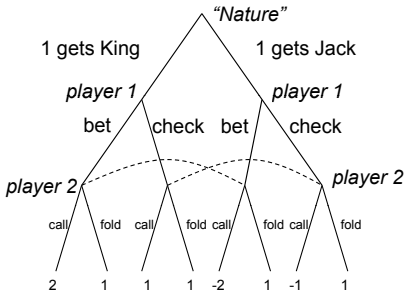

Figure 1: An extremely simple poker game.

(taking the expectation over Nature's randomness). This results in the following normal-form game:

	CC	CF	FC	FF
BB	0,0	0,0	1,-1	1,-1
BC	.5,-.5	1.5,-1.5	0,0	1,-1
CB	-.5,.5	-.5,.5	1,-1	1,-1
CC	0,0	1,-1	0,0	1,-1

While any extensive-form game can be converted to normal form in this manner, the representation size can blow up exponentially; we will return to this problem later.

Bayesian games

In recreational games, it is generally common knowledge how much each player values each outcome, because this is specified by the rules of the game. In contrast, when modeling real-world strategic situations, generally each player has some uncertainty over how much the other players value the different outcomes. This type of uncertainty is naturally modeled by *Bayesian games*. In a Bayesian game, each player draws a *type* from a distribution before playing the game. This type determines how much the player values each outcome.[1] Each player knows her own type, but not those of the other players. Again, Bayesian games can be converted to normal form, at the cost of an exponential increase in size.

Stochastic games

Stochastic games are simply a multiplayer generalization of Markov decision processes. In each state, all players choose an action, and the profile of actions selected determines the immediate rewards to all players as well as the transition probabilities to other states. A stochastic game with only one state is a *repeated game* (where the players play the same normal-form game over and over).

Compact representation schemes

One downside of the normal form is that its size is exponential in the number of players. Several representation schemes have been proposed to be able to represent "natural" games with many players compactly. Perhaps the best-known one is that of *graphical games* (Kearns, Littman, and Singh 2001), in which players are the nodes of a graph, and

[1] In general, a player's private information may also be relevant for determining how much another player values outcomes.

each player's utility is affected only by her own action and the actions of her neighbors. Another one is that of *action-graph games* (Jiang, Leyton-Brown, and Bhat 2011). In an action-graph game, the *actions* that players can take are the nodes of a graph, and a player's utility is affected only by the action she chose, and the number of players choosing each adjacent action.

Another compact representation scheme is that of MultiAgent Influence Diagrams (MAIDs) (Koller and Milch 2003). These are a generalization of Influence Diagrams, which in turn are a decision-theoretic generalization of Bayesian networks.

Solution Concepts

In this section, we discuss several standard solution concepts and how to compute their solutions. We focus primarily on two-player normal-form games.

Maximin/minimax strategies

Consider the following paranoid approach to playing a two-player zero-sum game: suppose player 1 assumes that player 2 will predict player 1's strategy perfectly, and respond to minimize player 1's utility (equivalently, to maximize his own utility). While this might seem like a relatively hopeless situation, we do allow player 1 to choose a *mixed* strategy, which is a probability distribution over pure strategies. We assume that player 2 can predict player 1's mixed strategy, but not the pure strategy that realizes from this distribution. Using Σ_i to denote player i's set of mixed strategies, player 1 can still guarantee that she receives at least her *maximin utility*, that is,

$$\max_{\sigma_1 \in \Sigma_1} \min_{s_2 \in S_2} u_1(\sigma_1, s_2)$$

Switching roles, it may be player 2 who adopts such a paranoid perspective. If so, he can guarantee that player 1 gets at most her *minimax utility*, that is,

$$\min_{\sigma_2 \in \Sigma_2} \max_{s_1 \in S_1} u_1(s_1, \sigma_2)$$

One might expect that this second situation is more advantageous to player 1. Amazingly, however, these two quantities are guaranteed to be the same! This is von Neumann's minimax theorem (von Neumann 1928), and it provides a very strong rationale for playing such a conservative strategy.

For example, consider the following zero-sum game.

	L	R
U	1,-1	-1,1
D	-2,2	1,-1

If the row player plays Up .6 of the time and Down .4 of the time, this guarantees her an expected utility of at least $-.2$. If the column player plays Left .4 of the time and Right .6 of the time, this guarantees him an expected utility of at least .2 (and thus guarantees the row player gets at most $-.2$).

As another example, in the poker game above, if player 1 plays BB $1/3$ and BC $2/3$ of the time, this guarantees her an expected utility of at least $1/3$. If player 2 plays CC $2/3$ and FC $1/3$ of the time, this guarantees him an expected utility of at least $-1/3$.

A maximin strategy of a normal-form game can be computed in polynomial time using a linear program (and the minimax theorem can be derived using this linear program, as a corollary of strong duality). Two-player zero-sum games can nevertheless be challenging to solve: a good example is that of heads-up (two-player) poker (say, Texas Hold'em). As pointed out above, converting such an extensive-form game to normal form results in an exponential increase in size. Fortunately, the *sequence form* (Romanovskii 1962; Koller and Megiddo 1992; von Stengel 1996) can be used to avoid this exponential increase while still allowing for a linear programming solution. Unfortunately, it is not even feasible to write down the entire extensive form! To address this, researchers have focused on techniques for shrinking a game tree into a smaller game tree that is close to or exactly equivalent (Billings et al. 2003; Gilpin and Sandholm 2007).

While many recreational games are zero-sum, most real-world strategic situations are not. Even in the security games that we will consider towards the end of this paper, which might seem to be the closest that we are likely to get to a truly adversarial situation in the real world, it is considered important to allow for the game not to be zero-sum (though, as we will discuss, these games share important properties with zero-sum games). Unfortunately, playing a maximin strategy is not necessarily a good idea in a general-sum game. For example, consider the following game:

	L	R
U	0,0	3,1
D	1,0	2,1

If the row player paranoidly believes that the column player is out to minimize her utility, then she should play Down, to guarantee herself a utility of 1. However, it is easy to see that the column player has every incentive to play Right, so that the row player is in fact better off playing Up.

Dominance and iterated dominance

Perhaps the strongest argument for *not* playing a strategy is that it is *strictly dominated*. We say that strategy σ_1 strictly dominates strategy σ_1' if it performs strictly better against every opponent strategy, that is, for all $s_2 \in S_2$, we have $u_1(\sigma_1, s_2) > u_1(\sigma_1', s_2)$.[2] A variant is *weak dominance*, where the strict inequality is replaced by a weak inequality, with the additional condition that the inequality be strict for at least one s_2. Of course, dominance is defined analogously for player 2. For example, in the previous game, Right strictly dominates Left. Based on this, we can remove Left from the game; after doing so, Up strictly dominates Down, and we can remove Down. This process of repeatedly removing dominated strategies is known as *iterated dominance*. As another example, in the poker game above, CB, CC, and CF are all weakly dominated, and FF strictly.

Whether a particular strategy is dominated (weakly or strongly) by a pure strategy can be computed straightforwardly by checking every pure strategy for that player. It is

[2] It does not matter if we replace "$s_2 \in S_2$" by "$\sigma_2 \in \Sigma_2$".

possible, though, that a strategy is not dominated by any pure strategy, but it is dominated by a mixed strategy. Whether this is the case can be checked using a linear program. This also allows us to efficiently perform iterated dominance: simply keep checking whether any pure strategies are dominated and remove them if they are, until nothing more can be removed. However, one may wonder whether the order in which we remove dominated strategies affects what remains in the end. It turns out that for iterated *strict* dominance it does not, but for iterated *weak* dominance it does—and because of this, in the latter case, problems such as determining whether a particular strategy can be removed through some sequence of eliminations are NP-hard. For more detail on computational aspects of (iterated) dominance, see (Gilboa, Kalai, and Zemel 1993; Conitzer and Sandholm 2005a; Brandt et al. 2011).

Nash equilibrium

A *Nash equilibrium* consists of a *profile* of strategies—one for each player—such that no player would want to deviate to another strategy individually, assuming that the others' strategies remain the same. That is, each player is playing a best response to the other players' strategies. For example, consider the following game between a speaker and an audience, where the former must decide whether to put effort into her talk (E) or not (NE), and the latter must decide whether to pay attention (A) or not (NA).

	A	NA
E	2,2	-1,0
NE	-7,-8	0,0

One Nash equilibrium is (A, E), in which both players are as happy as possible; another one is (NA, NE), which is worse for both players but no *individual* player would benefit from deviating; and a third, mixed-strategy equilibrium has the speaker playing E $4/5$ of the time and the audience playing A $1/10$ of the time (!), which leaves both players exactly indifferent between their choices (so that they do not mind randomizing) and is even worse for the speaker. A game may have no pure-strategy equilibria (for example, rock-paper-scissors), but at least one Nash equilibrium is guaranteed to exist in any finite game if we allow for mixed strategies (Nash 1950).

Regarding the computation of Nash equilibria, it is important to first precisely define the computational problem that we aim to solve. If our goal is to compute *all* Nash equilibria of a given game, the output may be exponential in length. A less ambitious goal is to compute just one—any one—Nash equilibrium of the game. Whether it is possible to do so in polynomial time was wide open for a significant period of time, leading Papadimitriou to declare settling the complexity of this problem "together with factoring [...] in my opinion the most important concrete open question on the boundary of P today" (Papadimitriou 2001). Eventually, it was shown that the problem is PPAD-complete with two players (Daskalakis, Goldberg, and Papadimitriou 2009; Chen, Deng, and Teng 2009) and FIXP-complete with three or more players (Etessami and Yannakakis 2010).

Depending on the circumstances, one may not even be satisfied to just compute any one equilibrium; for example, this may result in one of the "bad" equilibria in the above presentation game. Instead, one may wish to find the Nash equilibrium that maximizes player 1's utility, or the sum of the players' utilities. It turns out that such problems are NP-hard, and in fact inapproximable unless P=NP (Gilboa and Zemel 1989; Conitzer and Sandholm 2008).

In spite of these complexity results, researchers have proposed a variety of algorithms for computing Nash equilibria. For computing just one Nash equilibrium of a two-player game, a classic algorithm is the *Lemke-Howson* algorithm (Lemke and Howson 1964). While it is possible to construct families of games on which this algorithm's runtime is exponential (Savani and von Stengel 2006), it is usually quite fast. A simpler approach (Dickhaut and Kaplan 1991; Porter, Nudelman, and Shoham 2008) rests on the following observation. In any Nash equilibrium, some pure strategies will be optimal for a player to use and others, suboptimal, depending on the opponent's strategy. The optimal ones are the ones on which she can place positive probability herself. If it is known which of the pure strategies are optimal in this sense for each player for some equilibrium, then finding the equilibrium probabilities can be done in polynomial time. So, in a sense, determining which strategies get positive probability is the hard part. While there are exponentially many possibilities for which subset of strategies receives positive probability, we can nevertheless attempt to search through this space, and some heuristics, such as starting with the smallest subsets and working one's way up, and eliminating dominated strategies, turn out to be quite effective (Porter, Nudelman, and Shoham 2008). Perhaps unsurprisingly, families of games can be constructed on which such algorithms scale poorly.[3]

It is also possible to build on this technique to create a single mixed-integer program formulation for finding a Nash equilibrium. An advantage of this is that it allows for adding an objective function, so that one can compute the "best" Nash equilibrium (for example, one that maximizes player 1's utility, or the sum of the players' utilities). This approach significantly outperforms the alternative approach of first enumerating all equilibria with a version of one of the other algorithms and then choosing the best one (Sandholm, Gilpin, and Conitzer 2005).

[3] One such family is exemplified by the following game (Sandholm, Gilpin, and Conitzer 2005):

3,3	4,2	2,4	2,0	2,0	2,0	2,0
2,4	3,3	4,2	2,0	2,0	2,0	2,0
4,2	2,4	3,3	2,0	2,0	2,0	2,0
0,2	0,2	0,2	3,0	0,3	0,0	0,0
0,2	0,2	0,2	0,3	3,0	0,0	0,0
0,2	0,2	0,2	0,0	0,0	3,0	0,3
0,2	0,2	0,2	0,0	0,0	0,3	3,0

Games in this family have a unique Nash equilibrium where about half of the pure strategies are not in the supports, though none of them can be eliminated using dominance. They can, however, be eliminated using a slightly weaker notion (Conitzer and Sandholm 2005b).

Stackelberg strategies

For some games, if one player is able to *commit* to a strategy before the other player moves, this gives the former player a strategic advantage. The following game is often used to illustrate this point.

	L	R
U	1,1	3,0
D	0,0	2,1

This game is solvable by iterated strict dominance: Up dominates Down, and after removing Down, Left dominates Right, suggesting that the outcome will be (Up, Left) with utilities $(1, 1)$. However, now suppose that we change the rules of the game a bit, so that the row player is able to credibly *commit* to a strategy before the column player moves. It can be seen that the row player is better off committing to Down instead: this incentivizes the column player to play Right, resulting in a utility of 2 for player 1. Even better would be to commit to play Down with probability $.51$. This still incentivizes player 2 to play Right, but now player 1 gets 3 exactly $.49$ of the time, resulting in an expected utility of 2.49. It is easy to see that this can be pushed up as far as 2.5, which is optimal. The corresponding strategy for player 1 is called a *Stackelberg mixed strategy*.

Intriguingly, in contrast to the hardness results for computing Nash equilibria, for a normal-form game a Stackelberg mixed strategy can be computed in polynomial time using linear programming (Conitzer and Sandholm 2006; von Stengel and Zamir 2010; Conitzer and Korzhyk 2011). Unfortunately, computing Stackelberg mixed strategies in Bayesian games is NP-hard (Conitzer and Sandholm 2006) (and the optimal utility for the leader is inapproximable unless P=NP (Letchford, Conitzer, and Munagala 2009)). Nevertheless, techniques from mixed integer programming and branch-and-bound search have been used to obtain algorithms for computing Stackelberg strategies in Bayesian games that scale reasonably well (Paruchuri et al. 2008; Jain, Kiekintveld, and Tambe 2011), and such an algorithm underlies the LAX application discussed below. The computation of Stackelberg strategies has also been studied in extensive-form games (Letchford and Conitzer 2010) and stochastic games (Letchford et al. 2012).

Application to Security Domains

There are several reasons that it is natural to model security domains using game theory. They clearly involve multiple parties with very different (though not necessarily completely opposed) interests. The idea of playing a mixed strategy is also natural: for example, security personnel want their patrolling strategies to be unpredictable. Still, there is a large gap between making such high-level observations and deploying a system that directly computes a game-theoretic solution in a high-stakes real-world security application. In recent years, the TEAMCORE group at the University of Southern California has done precisely this, in several applications. One is airport security: they introduced the ARMOR system at Los Angeles International Airport (LAX) to schedule checkpoints on roads entering the airport, as well as canine units (Pita et al. 2008); their later GUARDS system is being evaluated for national deployment (Pita et al. 2011). Their IRIS system schedules Federal Air Marshals to flights (Tsai et al. 2009). They have also started work with the US Coast Guard: their PROTECT system is deployed in Boston's port to randomize patrols (Shieh et al. 2012).

The Stackelberg model underlies every one of these applications.[4] The typical argument for playing a Stackelberg mixed strategy in this context is as follows. Every period, the defender (e.g., security personnel) must take a course of action, such as allocating scarce security resources to locations. The attacker, however, does not have to attack every period; instead, he can just observe the defender's action a number of times, for example driving to an airport terminal to see whether there is a checkpoint, without carrying anything suspect. This way, he can learn the probability distribution over where the checkpoints are, before finally choosing a plan of attack. Whether this argument is accurate in all of these security applications is debatable, and some work has addressed the case where there is explicit uncertainty about whether the attacker can observe the mixed strategy (Korzhyk et al. 2011; Korzhyk, Conitzer, and Parr 2011b).

The structure of security games

It is not clear that the full generality of game theory is needed to address security games; these games may have special structure that allows us to avoid some of the difficulties that game theory faces in general. But what is the defining structure that makes a game a security game? While it appears unlikely that the community will soon settle on a single definition of security games that captures all applications that we may wish to address in this space, nevertheless at least one fairly general definition has been proposed (Kiekintveld et al. 2009). Under this definition, a security game consists of:

- a set of *targets*, of which the attacker will choose one to attack;
- a set of *resources* for the defender;
- a set of *schedules*, each of which consists of a subset of the targets, to which a resource may be assigned, possibly with constraints on which resources can be assigned to which schedules;
- for each target, two utilities for each of the defender and the attacker—one for the case where this target is attacked and at least one resource is assigned to it, and one for the case where it is attacked and no resource is assigned to it.

For example, in the Federal Air Marshals problem, targets are flights, resources are FAMs, and schedules are tours of multiple flights to which FAMs may be assigned.

It turns out that in such games, the Nash equilibria are interchangeable, meaning that miscoordination about which

[4]The PROTECT system uses a quantal response model for the attacker's behavior rather than assuming perfect rationality; an optimal strategy in this framework is computed using the PASAQ (Yang, Ordóñez, and Tambe 2012) algorithm.

equilibrium to play is impossible; also, under a minor assumption, a Stackelberg strategy for the defender is guaranteed to also be a Nash strategy—though this no longer holds if the attacker also has multiple resources (Korzhyk et al. 2011). The interchangeability property still holds in a version with multiple attacker resources (Korzhyk, Conitzer, and Parr 2011a).

The above definition also provides a compact representation language for security games. Indeed, a pure strategy for the defender is an allocation of resources to schedules, and there are exponentially many such allocations. Therefore, an efficient algorithm for security games must operate directly on the security game representation, rather than on the normal form. A number of algorithmic and complexity results have been derived for the problems of computing Stackelberg and Nash solutions of these games (Kiekintveld et al. 2009; Korzhyk, Conitzer, and Parr 2010; Jain et al. 2010; Korzhyk, Conitzer, and Parr 2011a).

Extensions of security games

While the security game representation from the previous subsection elegantly captures many domains of interest, there are exceptions. One example is the "protecting a city" problem (Tsai et al. 2010). This problem is motivated by the attacks on Mumbai in 2008, where terrorists arrived in Mumbai by speedboats and traveled through the city to a variety of targets. The Mumbai police subsequently started setting up checkpoints on roads to prevent such attacks.

The game is modeled as a graph whose edges represent roads, and some of whose nodes represent targets or entry points into the graph. The defender chooses a subset of at most k edges to defend, and the attacker chooses a path from an entry point to a target (resulting in an exponentially large attacker strategy space). Even if the game is modeled as a zero-sum game in which the targets have varying values, it turns out to be quite difficult to scale. The state-of-the-art approach is a double-oracle algorithm in which strategies are generated incrementally, but improvements in scalability are still needed to be able to scale to (for example) the full graph of Mumbai (Jain et al. 2011).

Conclusion

The proliferation of deployed security applications in the last few years suggests that this line of work may have much more impact yet. It is not difficult to imagine that these applications may gradually extend beyond security domains. It is also important to point out that not all of the game-theoretic work (or, more generally, the "economic paradigms" work) in the multiagent systems community concerns the problem of computing solutions of a given game. For example, in mechanism design, the goal is to *design* the game in a way that results in good outcomes when it is played by strategic agents. The boundaries between these lines of work are still blurry and fluid, considering, for example, recent work on providing incentives for agents to act in a desired manner in a given domain (Monderer and Tennenholtz 2004; Zhang, Chen, and Parkes 2009; Anderson, Shoham, and Altman 2010). This leaves this author hopeful that some surprising new applications have yet to emerge!

Acknowledgments

I thank NSF Awards IIS-0812113, IIS-0953756, and CCF-1101659, as well as ARO MURI Grant W911NF-11-1-0332 and an Alfred P. Sloan fellowship, for support.

References

Anderson, A.; Shoham, Y.; and Altman, A. 2010. Internal implementation. *AAMAS*, 191–198.

Billings, D.; Burch, N.; Davidson, A.; Holte, R.; Schaeffer, J.; Schauenberg, T.; and Szafron, D. 2003. Approximating game-theoretic optimal strategies for full-scale poker. *IJCAI*, 661–668.

Brandt, F.; Brill, M.; Fischer, F.; and Harrenstein, P. 2011. On the complexity of iterated weak dominance in constant-sum games. *Theory of Computing Systems* 49(1):162–181.

Chen, X.; Deng, X.; and Teng, S.-H. 2009. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM* 56(3).

Conitzer, V., and Korzhyk, D. 2011. Commitment to correlated strategies. *AAAI*, 632–637.

Conitzer, V., and Sandholm, T. 2005a. Complexity of (iterated) dominance. *EC*, 88–97.

Conitzer, V., and Sandholm, T. 2005b. A generalized strategy eliminability criterion and computational methods for applying it. *AAAI*, 483–488.

Conitzer, V., and Sandholm, T. 2006. Computing the optimal strategy to commit to. *EC*, 82–90.

Conitzer, V., and Sandholm, T. 2008. New complexity results about Nash equilibria. *Games and Economic Behavior* 63(2):621–641.

Daskalakis, C.; Goldberg, P.; and Papadimitriou, C. H. 2009. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing* 39(1):195–259.

Dickhaut, J., and Kaplan, T. 1991. A program for finding Nash equilibria. *The Mathematica Journal* 87–93.

Etessami, K., and Yannakakis, M. 2010. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing* 39(6):2531–2597.

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

Gilboa, I.; Kalai, E.; and Zemel, E. 1993. The complexity of eliminating dominated strategies. *Mathematics of Operation Research* 18:553–565.

Gilpin, A., and Sandholm, T. 2007. Lossless abstraction of imperfect information games. *Journal of the ACM* 54(5).

Jain, M.; Kardes, E.; Kiekintveld, C.; Ordóñez, F.; and Tambe, M. 2010. Security games with arbitrary schedules: A branch and price approach. *AAAI*, 792–797.

Jain, M.; Korzhyk, D.; Vanek, O.; Conitzer, V.; Pechoucek, M.; and Tambe, M. 2011. A double oracle algorithm for zero-sum security games on graphs. *AAMAS*, 327–334.

Jain, M.; Kiekintveld, C.; and Tambe, M. 2011. Quality-bounded solutions for finite Bayesian Stackelberg games: Scaling up. *AAMAS*, 997–1004.

Jiang, A. X.; Leyton-Brown, K.; and Bhat, N. A. R. 2011. Action-graph games. *Games and Economic Behavior* 71(1):141–173.

Kearns, M.; Littman, M.; and Singh, S. 2001. Graphical models for game theory. *UAI*, 253–260.

Kiekintveld, C.; Jain, M.; Tsai, J.; Pita, J.; Ordóñez, F.; and Tambe, M. 2009. Computing optimal randomized resource allocations for massive security games. *AAMAS*, 689–696.

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

Koller, D., and Milch, B. 2003. Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior* 45(1):181–221.

Korzhyk, D.; Yin, Z.; Kiekintveld, C.; Conitzer, V.; and Tambe, M. 2011. Stackelberg vs. Nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *JAIR* 41(2):297–327.

Korzhyk, D.; Conitzer, V.; and Parr, R. 2010. Complexity of computing optimal Stackelberg strategies in security resource allocation games. *AAAI*, 805–810.

Korzhyk, D.; Conitzer, V.; and Parr, R. 2011a. Security games with multiple attacker resources. *IJCAI*, 273–279.

Korzhyk, D.; Conitzer, V.; and Parr, R. 2011b. Solving Stackelberg games with uncertain observability. *AAMAS*, 1013–1020.

Lemke, C., and Howson, J. 1964. Equilibrium points of bimatrix games. *Journal of the Society of Industrial and Applied Mathematics* 12:413–423.

Letchford, J., and Conitzer, V. 2010. Computing optimal strategies to commit to in extensive-form games. *EC*, 83–92.

Letchford, J.; MacDermed, L.; Conitzer, V.; Parr, R.; and Isbell, C. 2012. Computing optimal strategies to commit to in stochastic games. *AAAI*.

Letchford, J.; Conitzer, V.; and Munagala, K. 2009. Learning and approximating the optimal strategy to commit to. *SAGT*, 250–262.

Monderer, D., and Tennenholtz, M. 2004. K-Implementation. *JAIR* 21:37–62.

Nash, J. 1950. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences* 36:48–49.

Papadimitriou, C. H. 2001. Algorithms, games and the Internet. *STOC*, 749–753.

Paruchuri, P.; Pearce, J. P.; Marecki, J.; Tambe, M.; Ordóñez, F.; and Kraus, S. 2008. Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games. *AAMAS*, 895–902.

Pita, J.; Jain, M.; Western, C.; Portway, C.; Tambe, M.; Ordóñez, F.; Kraus, S.; and Parachuri, P. 2008. Deployed ARMOR protection: The application of a game-theoretic model for security at the Los Angeles International Airport. *AAMAS*, 125–132.

Pita, J.; Tambe, M.; Kiekintveld, C.; Cullen, S.; and Steigerwald, E. 2011. GUARDS - Game theoretic security allocation on a national scale. *AAMAS*, 37–44.

Porter, R.; Nudelman, E.; and Shoham, Y. 2008. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior* 63(2):642–662.

Romanovskii, I. 1962. Reduction of a game with complete memory to a matrix game. *Soviet Mathematics* 3:678–681.

Sandholm, T.; Gilpin, A.; and Conitzer, V. 2005. Mixed-integer programming methods for finding Nash equilibria. *AAAI*, 495–501.

Savani, R., and von Stengel, B. 2006. Hard-to-solve bimatrix games. *Econometrica* 74:397–429.

Shieh, E.; An, B.; Yang, R.; Tambe, M.; Baldwin, C.; DiRenzo, J.; Maule, B.; and Meyer, G. 2012. PROTECT: A deployed game theoretic system to protect the ports of the United States. *AAMAS*.

Tsai, J.; Rathi, S.; Kiekintveld, C.; Ordóñez, F.; and Tambe, M. 2009. IRIS - a tool for strategic security allocation in transportation networks. *AAMAS*, 37–44.

Tsai, J.; Yin, Z.; young Kwak, J.; Kempe, D.; Kiekintveld, C.; and Tambe, M. 2010. Urban security: Game-theoretic resource allocation in networked physical domains. *AAAI*, 881–886.

von Neumann, J. 1928. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen* 100:295–320.

von Stengel, B., and Zamir, S. 2010. Leadership games with convex strategy sets. *Games and Economic Behavior* 69:446–457.

von Stengel, B. 1996. Efficient computation of behavior strategies. *Games and Economic Behavior* 14(2):220–246.

Yang, R.; Ordóñez, F.; and Tambe, M. 2012. Computing optimal strategy against quantal response in security games. *AAMAS*.

Zhang, H.; Chen, Y.; and Parkes, D. C. 2009. A general approach to environment design with one agent. *IJCAI*, 2002–2014.