Fast Equilibrium Computation for Infinitely Repeated Games

Garrett Andersen
Department of Computer Science
Duke University
Durham, NC 27708, USA
garrett@cs.duke.edu

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

Abstract

It is known that an equilibrium of an infinitely repeated two-player game (with limit average payoffs) can be computed in polynomial time, as follows: according to the folk theorem, we compute minimax strategies for both players to calculate the punishment values, and subsequently find a mixture over outcomes that exceeds these punishment values. However, for very large games, even computing minimax strategies can be prohibitive. In this paper, we propose an algorithmic framework for computing equilibria of repeated games that does not require linear programming and that does not necessarily need to inspect all payoffs of the game. This algorithm necessarily sometimes fails to compute an equilibrium, but we mathematically demonstrate that most of the time it succeeds quickly on uniformly random games, and experimentally demonstrate this for other classes of games. This also holds for games with more than two players, for which no efficient general algorithms are known.

Introduction

The computation of game-theoretic equilibria is a topic that has long been of interest to the AI community, with applications such as computer poker as well as in security domains. Computing an equilibrium of a two-player zero-sum game (which must consist of maximin/minimax strategies for both players) is equivalent to linear programming (Dantzig 1951; Adler forthcoming) and can hence be done in polynomial time. On the other hand, computing a Nash equilibrium of a two-player general-sum game is PPAD-complete (Daskalakis, Goldberg, and Papadimitriou 2009; Chen, Deng, and Teng 2009) (and this is FIXP-complete for games with three or more players (Etessami and Yannakakis 2010)).

In reality, it is rare to have a truly single-shot game with players that have never interacted previously and never will again. On perhaps the other extreme of the modeling spectrum, in an infinitely repeated game, the same game is played infinitely many times. Future payoffs may be discounted, or agents may care about their limit average payoff (defined later in the paper); we focus on the latter here. In this context, the famous *folk theorem* characterizes the vectors of limit average payoffs that can be obtained in equilibria of two-player games as follows: (1) *enforceability*: every player must obtain at least her maximin value, and (2) *feasibility*: the vector of payoffs must be a convex combination of outcomes in the game. It has been observed that this characterization also allows one to *compute* such an equilibrium in polynomial time (Littman and Stone 2005). This algorithm requires the computation of minimax strategies for both players.

On sufficiently large games, however, even the computation of a minimax strategy can be challenging or even prohibitive. For example, it is known to be challenging to compute minimax strategies in certain hider-seeker games (Halvorson, Conitzer, and Parr 2009) or security games on graphs (Jain et al. 2011). Perhaps more interestingly, if one thinks of the exceptionally complex game of real life, it is unimaginable that we could even specify all the payoffs of this game, let alone compute minimax strategies. But we somehow manage to get on in real life. One might argue that this is due to a perhaps unappreciated aspect of the folk theorem: while we can maximially punish a deviating player by playing a minimax strategy, such draconian punishment is generally not necessary. All that is really needed is a punishment strategy that is sufficiently harsh to make the player regret having deviated.[1]

In this paper, we take advantage of this insight to save ourselves from expensive minimax computations. Rather, we attempt to find punishment strategies that have a reasonably high deterrent effect, as well as to find feasible profiles that are reasonably desirable. Once the utilities corresponding to the latter (weakly) exceed those corresponding to the former, we have an equilibrium. Specifically, to punish a specific player, we randomly select a small set of strategies for each other player, and have each of them randomize uniformly over these. From this, we can compute the best-response value for the punished player to this punishment profile. Once we have these values for each player, we scan through outcomes of the game until we find one that has all its payoffs greater than or equal to these punishment values

[1] Indeed, in practice, we like for the punishment to fit the crime, rather than (to be a bit dramatic) to impose the death penalty on jaywalkers.

(so that no player would want to deviate from this outcome and get punished instead). We note that this algorithm does not necessarily need to inspect every payoff of the game, which is particularly valuable if there is significant deliberation cost to determining each individual payoff.

This algorithm will not always succeed in finding an equilibrium. For example, if we have an infinitely repeated two-player *zero-sum* game, each player must obtain exactly the value of the game on the path of play, or otherwise one of them would deviate to a maximin strategy. But, in a random two-player zero-sum game, it is unlikely that both (a) there is a single outcome of the game that gives each player exactly the value, and (b) there are minimax strategies that uniformly randomize over a subset of the pure strategies. On other families of games, however, our algorithm does quite well, leading to the perhaps surprising conclusion that in this case, zero-sum games are the hardest ones!

Another benefit of our approach is that it also (often) works on games with three or more players. The algorithm based on linear programming does not work for this setting: when two agents are attempting to jointly punish a third agent as much as possible, they actually face an NP-hard problem (Borgs et al. 2010) (see also (Hansen et al. 2008)). Indeed, it has been shown that computing a Nash equilibrium of a three-player infinitely-repeated game is PPAD-hard (Borgs et al. 2010). This is assuming that the punishing players must each pick a mixed strategy, i.e., they cannot correlate their punishing actions; if they can, then the problem can indeed be solved with linear programming (Kontogiannis and Spirakis 2008). However, when our approach is successful, the punishing players will play mixed strategies.

Background

An m-player normal-form game is defined by a (finite) set of pure strategies S_i for every player $i \in \{1, \ldots, m\}$, and, for each player i, a payoff function $u_i : S \to \mathbb{R}$, where $S = S_1 \times \ldots \times S_m$. For simplicity, we focus on $n \times \ldots \times n$ games, i.e., $|S_i| = n$ for all i. Consider now a sequence $(s^0, s^1, \ldots) \in S^\infty$ of play in the infinitely repeated game. We assume that player i attempts to maximize i's limit average payoff, that is,

$$\liminf_{T\to\infty} E[(1/T)\sum_{t=0}^{T-1} u_i(s^t)]$$

(It is standard to put in the $\inf$ to ensure the limit is well defined.) Let σ_i denote a mixed strategy for i, i.e., a probability distribution over S_i, and let $\sigma_{-i} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_m)$ denote a mixed strategy profile for the players other than i. Player i's *minimax value* is $v_i = \min_{\sigma_{-i}} \max_i u_i(\sigma_i, \sigma_{-i})$. A payoff profile $(a_1, \ldots, a_m)$ is said to be *enforceable* if $a_i \geq v_i$ for all i; it is said to be *feasible* if there exists a distribution $p : S \to [0, 1]$ over S such that for all i, $\sum_{s\in S} p(s)u_i(s) = a_i$.

Theorem 1 (Folk Theorem) *If a payoff profile $(a_1, \ldots, a_m)$ is enforceable and feasible, then it is attained in a Nash equilibrium of the game.*

The intuition is simple: by the feasibility condition, we can create an infinite sequence of strategy profiles that will give limit average payoffs $(a_1, \ldots, a_m)$. Every player j's strategy is to follow that sequence, unless some player i has deviated, in which case j will play according to the strategy profile σ_{-i} that guarantees that i gets at most v_i forever. Because the latter situation results (at best) in limit average payoff $v_i \leq a_i$ for player i, i has no incentive to deviate.

For the case of a two-player game, it has been previously shown that the folk theorem can be used to compute such an equilibrium in polynomial time (Littman and Stone 2005). The most computationally demanding part of this algorithm consists of computing each player's minimax strategy (to determine the optimal way to punish the other player), which corresponds to a linear programming problem. With three or more players, this approach fails because computing the joint minimax strategy σ_{-i} is NP-hard (Borgs et al. 2010).

Algorithm

In this section, we present the basic version of our algorithm. In the next section, we analyze this version for uniformly random games. We then consider a variant of this algorithm in the experimental section after that.

As mentioned above, the key idea of our algorithm is to quickly find punishment strategies that have a reasonable (though not necessarily optimal) deterrent effect. We do so by randomly picking k pure strategies for each player and considering the strategy that mixes uniformly over these k strategies. Subsequently, we can scan through the outcomes to find one that has all players' payoffs greater than or equal to the punishment values.

Algorithm 1
for each $i, j \in \{1, \ldots, m\}$ **with** $i \neq j$
  $P_k \leftarrow \text{RandomSample}(S_i, k)$
  $\sigma_i^j \leftarrow \text{Uniform}(P_k)$
for each $j \in \{1, \ldots, m\}$
  $w_j \leftarrow \max_{s_j \in S_j} u_j(s_j, \sigma_{-j}^j)$
$S_{\text{accessed}} \leftarrow \{s \in S : |\{i \in \{1, \ldots, m\} : s_i \notin P_i\}| \leq 1\}$
for each $s \in S$ **starting with** $s \in S \setminus S_{\text{accessed}}$
  if for all $i \in \{1, \ldots, m\}$, $u_i(s) \geq w_i$
  then return $(s; \sigma_{-1}^1, \ldots, \sigma_{-m}^m)$
return failure

Here, $\text{RandomSample}(S_i, k)$ returns a random subset of S_i of size k; $\text{Uniform}(P_k)$ returns the uniform strategy over P_k; σ_i^j is the mixed strategy that i uses to punish j; $\sigma_{-j}^j = (\sigma_1^j, \ldots, \sigma_{j-1}^j, \sigma_{j+1}^j, \ldots, \sigma_m^j)$ is the profile of strategies punishing j; w_j is the *punishment value* for j, i.e., the highest utility j can obtain when the others are punishing j; S_{accessed} is the set of pure strategy profiles whose payoffs the algorithm accessed while determining the punishment values (these entries will not be visited in the "exploration" phase because they would be correlated with the w_j, which complicates analysis); and $(s; \sigma_{-1}^1, \ldots, \sigma_{-m}^m)$ denotes the strategy profile where everyone plays according to s, unless player j has deviated, in which case player i plays σ_i^j. (If

multiple players have deviated, the players can play arbitrarily.)

Proposition 1 *If Algorithm 1 does not return failure, then it returns a Nash equilibrium of the infinitely repeated game with limit-average utilities.*

Proof. If player i does not deviate, i will obtain a limit-average utility of $u_i(s)$. If player i does deviate at some point, this will result in all other players playing σ^i_{-i}, and player i will be able to obtain a limit-average utility of at most w_i. Because $w_i \leq u_i(s)$, there is no incentive to deviate. □

This algorithm only tries one random selection of punishment strategies, and if it fails, it gives up. A natural variant is to redraw the punishment strategies in case of failure. We discuss such a variant in the experimental section, but in the next, theoretical section, we do not consider such redrawing.

Theoretical results for uniformly random games

In this section, we theoretically analyze how the algorithm performs on games where all the payoffs are drawn independently from the uniform distribution $U[0,1]$. We first give some intuition about why one might expect the algorithm to perform well here, and then proceed with a formal analysis.

Intuition

In a two-player game, if the row player's mixed strategy is to play uniformly at random over a given k rows (say, the first k rows), then consider the expected utility η_1 of the column player for playing the first column of the game. η_1 will be different depending on the game; because the game is drawn at random, η_1 is itself a random variable. Its expectation is $1/2$. As k increases, η_1's probability distribution will quickly become concentrated around $1/2$, by the law of large numbers. In fact, if we let η_j denote the column player's expected utility for playing the jth column, even $\max_j \eta_j$ becomes unlikely to be far away from $1/2$ as k increases. But $\max_j \eta_j$ is the column player's best-response utility against the row player's strategy. Thus, if k is large enough, then with high probability, the row player playing uniformly randomly over the first k rows (or a randomly selected set of k rows) results in a punishment value for the column player that is close to $1/2$. The same will be true when we switch the roles of the row and column players.

Now, for any entry of the game, there is a probability of $1/4$ that both players' utilities in this entry exceed $1/2$, and a somewhat smaller probability that both players' utilities in this entry exceed their punishment values. To be precise, we should worry about the correlation between the entries' utilities and the punishment values—but if k is significantly smaller than the number of rows/columns, then plenty of the entries have not even been considered in the punishment value calculation and are therefore independent of them. Thus, if we scan through these entries one at a time, we would expect to relatively quickly find one that exceeds the punishment value for both players, and we will have found a Nash equilibrium of the infinitely repeated game.

Formal analysis

Besides being informal, the above does not give much insight into how large k needs to be, and how all this varies with the size of the game. We now provide a more formal analysis. To do so, we first prove two lemmas. The first bounds the probability that some of the punishment values are high (i.e., not all of the punishment strategies are effective). The second bounds the probability that we fail to find an outcome all of whose payoffs are greater than or equal to the punishment values.

For the first bound, we first review the special case of Hoeffding's concentration bound that we use.

Theorem 2 (Hoeffding's inequality) *Let $X_1, ..., X_c$ be independent random variables that are bounded between 0 and 1, and let $\overline{X} = (\sum_{i=1}^{c} X_i)/c$ be their average. Then the following inequality holds:*

$$Pr(\overline{X} - E[\overline{X}] \geq t) \leq e^{-2ct^2}$$

We are now ready to present our bound on the probability that at least one of the punishment strategies does not have a significant deterrent effect.

Lemma 1 *In $\underbrace{n \times \ldots \times n}_{m}$ games where every payoff is drawn independently from $U[0,1]$, if every punishment strategy σ^j_i is the uniform distribution over k pure strategies, then the probability that all players have punishment values of at most w is at least*

$$1 - mne^{-2k^{m-1}(w-1/2)^2}$$

Proof. Suppose player j is being punished, so that the rest of the players will each play uniformly randomly over k strategies. Then, for each pure strategy s_j that j may play, there are k^{m-1} equally likely outcomes of the game. The expected utility $u_j(s_j, \sigma^j_{-j})$ that j experiences as a result is the average of k^{m-1} independent random variables drawn from $U[0,1]$. The Hoeffding bound implies:

$$Pr(u_j(s_j, \sigma^j_{-j}) - 1/2 \geq t) \leq e^{-2k^{m-1}t^2}$$

Player j has n actions to choose from in response to the punishment, so by the union bound, the probability that player j has a punishment value larger than $\frac{1}{2} + t$ is at most:

$$Pr(\max_{s_j \in S_j} \{u_j(s_j, \sigma^j_{-j})\} - 1/2 \geq t) \leq ne^{-2k^{m-1}t^2}$$

There are m players, so by the union bound again, the probability that some player has a punishment value larger than $\frac{1}{2} + t$ is at most:

$$Pr(\max_{j \in \{1,\ldots,m\}, s_j \in S_j} \{u_j(s_j, \sigma^j_{-j})\} - 1/2 \geq t) \leq$$

$$mne^{-2k^{m-1}t^2}$$

The lemma follows from considering the complementary event and letting $t = w - 1/2$. □

Next, we give our bound on the probability that we fail to find an outcome of the game that (weakly) exceeds the punishment values. This bound is very simple, as long as we restrict ourselves to only considering outcomes that have not been used to compute the punishment values and are therefore uncorrelated with the punishment values. (In general, there will be plenty of these, because at most mnk^{m-1} outcomes are used to compute the punishment values, which is much smaller than the total number of outcomes n^m as long as k is not too large.)

Lemma 2 *In* $\underbrace{n \times \ldots \times n}_{m}$ *games where every payoff is drawn independently from* $U[0,1]$*, if the punishment value for every player is at most* w*, and we consider* x *outcomes of the game that are uncorrelated with the punishment values, then there is a probability of at most* $(1-(1-w)^m)^x$ *that we fail to find an outcome such that all of its payoffs exceed the punishment values.*

Proof. Any outcome of the game that is uncorrelated with the punishment values has a probability of at least $(1-w)^m$ of exceeding all the punishment values. Thus, the probability that this is not true for all x outcomes is at most $(1-(1-w)^m)^x$. □

Theorem 3 *For* $\epsilon > 0$*, if there exist positive integers* k *and* x *with* $x \leq n^m - mnk^{m-1}$ *such that*

$$\sqrt{\frac{log(\frac{mn}{\epsilon})}{2k^{m-1}}} + \sqrt[m]{1-\sqrt[x]{\epsilon}} \leq 1/2$$

then Algorithm 1 using value k *will, with probability at least* $1-2\epsilon$*, return an equilibrium after accessing at most* $mnk^{m-1}+xm$ *payoffs of the game.*

Proof. Set $w = 1/2 + \sqrt{\frac{log(\frac{mn}{\epsilon})}{2k^{m-1}}}$, so that $1 - mne^{-2k^{m-1}(w-1/2)^2} = 1-\epsilon$. Then, by Lemma 1, if we use Algorithm 1 with value k then there is a probability of at most ϵ that there is a player with a punishment value greater than w. Additionally, by the inequality in the theorem, $w = 1/2 + \sqrt{\frac{log(\frac{mn}{\epsilon})}{2k^{m-1}}} \leq 1 - \sqrt[m]{1-\sqrt[x]{\epsilon}}$. Let us say $w' = 1 - \sqrt[m]{1-\sqrt[x]{\epsilon}}$, so that $(1-(1-w')^m)^x = \epsilon$. We have $w' \geq w$, so that if w is greater than or equal to all the punishment values, then so is w'. And, by Lemma 2, if all the punishment values are at most w', there is a probability of at most ϵ that after considering x outcomes that are uncorrelated with the punishment values (and there exists x such outcomes because $x \leq n^m - mnk^{m-1}$) we fail to find an outcome such that all of its payoffs exceed the punishment values. Thus, the probability of the algorithm failing is at most $\epsilon + (1-\epsilon)\epsilon \leq 2\epsilon$. □

Setting the parameters optimally

Theorem 3 gives us sufficient conditions on k and x for Algorithm 1 to have a $1-2\epsilon$ probability of success after accessing $mnk^{m-1}+xm$ payoffs of the game. Taking the number of payoffs accessed as a proxy for runtime, this leads to the following natural optimization problem: for given m, n, and desired error probability 2ϵ, choose k and x to minimize $mnk^{m-1}+xm$ under the constraint of having at most 2ϵ probability of failing. Theorem 3 allows us to formulate this as the following simple optimization problem.

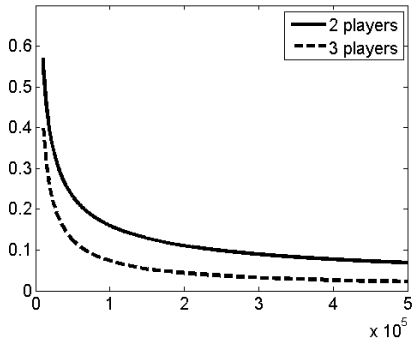

Figure 1: Fraction of the total number of payoffs that must be accessed by the algorithm to get a 99% chance of success, as a function of the total number of payoffs (mn^m) in the game, according to Theorem 3 for the optimal values of k and x.

minimize $mnk^{m-1}+xm$
subject to

$$\sqrt{\frac{log(\frac{mn}{\epsilon})}{2k^{m-1}}} + \sqrt[m]{1-\sqrt[x]{\epsilon}} \leq 1/2$$

Figure 1 shows the number of payoffs in the game that Algorithm 1 needs to access to succeed with probability $2\epsilon = .01$, for the optimal values of k and x calculated according to the above optimization problem. The x-axis shows the total number of payoffs in the game. As can be seen, the fraction of the payoffs that needs to be accessed is sublinear.[2]

Experiments

The theoretical results in the previous section only concern games in which all the payoffs are drawn i.i.d. from $U[0,1]$. How does our approach fare on other classes of games? In this section, we study this question experimentally, using the GAMUT family of game generators (Nudelman et al. 2004).

The algorithm we use here is different from Algorithm 1 in a few ways. Most notably, instead of drawing the punishment strategies only once, we resample them periodically, after scanning through a number of outcomes and failing to

[2]The graph is cut off at the left end before going up to 100% because we choose to ignore a certain portion of the game in order to simplify analysis. As mentioned previously, any outcome of the game used in the computation of the players' punishment values will not be accessed again in the "exploration" phase in order to not have to deal with correlation. Notice, however, that when an outcome of the game is accessed in the punishment phase, the punishing players' utilities are not relevant. Thus, these utilities are never accessed, which is why we are never able to use 100% of the utilities in the game.

find one that meets or exceeds all the punishment values. Specifically, we do the resampling so that the number of payoffs that we consider in the scanning phase is is half of the number of payoffs that we consider when calculating punishment values (which is mnk^{m-1} payoffs per punishment phase). We keep the best punishment value found for each player so far. This significantly improves the performance of the algorithm, because some game families require quite specific punishment strategies that one would be unlikely to find sampling only once. In each scanning phase, we randomly choose the next outcome to scan, and unlike in Algorithm 1, we do not keep track of which outcomes have been accessed before.

In order to compare to the LP-based algorithm, we first need to address the previously mentioned issue that our algorithm is not complete: for example, in 2-player zero-sum games, generally there will not be any equilibrium that can be discovered by our algorithm. To address this, we can imagine running the LP-based algorithm in parallel to ours. If one of these two algorithms finishes first after t seconds, then $2t$ total seconds of work will have been done (possibly across multiple machines). Because the LP-based algorithm is complete, so is the algorithm that runs both in parallel. In the experiments we run below, this is how we count the runtime of our algorithm: imagining the LP-based algorithm running in parallel, and charging for its runtime as well (until a solution has been found). This directly implies that our algorithm can never take more than twice the runtime of the LP-based approach: the worst case is when the LP-based approach is always faster, in which case our approach has twice the cost.

Figure 2 shows the results on 2-player games for various values of k, compared to the pure LP-based approach. Our algorithm performs better on RandomGame (as expected per the theoretical results), DispersionGame, RandomGraphicalGame, MinimumEffortGame, PolymatrixGame, CovariantGame, TravelersDilemma, and WarOfAttrition. It performs worse on BertrandOligopoly, MajorityVoting, UniformLEG, BidirectionalLEG, GrabTheDollar, RandomLEG, CournotDuopoly, GuessTwoThirdsAve, RandomZeroSum, and LocationGame. (Some of these game families are (close to) zero-sum, so it is not surprising that our algorithm fails.)

Figure 3 shows the results on 3-player games for various values of k, compared to the pure LP-based approach. The LP-based approach, in this case, allows for punishing players to correlate their strategies, because otherwise the problem is known to be hard. There are fewer game classes here because some of the GAMUT game classes do not extend to three players. The results are similar to the 2-player case.

When considering games with more than 2 players, one advantage of our algorithm is that it produces equilibria that do not require punishing players to be able to correlate (unless the LP-based approach finishes faster and we use its solution). Of course, some of the time, the LP-based approach also happens to return an equilibrium that does not require correlated punishment. Figure 4 shows how often the algorithms return an equilibrium that does not require correlated punishment. Regardless of k or the size of the game, our algorithm (with the LP-based algorithm in parallel) returns such a strategy more than 80% of the time, whereas the LP-based approach does so about 40% of the time. Whether each algorithm returns such a strategy depends primarily on the game class.

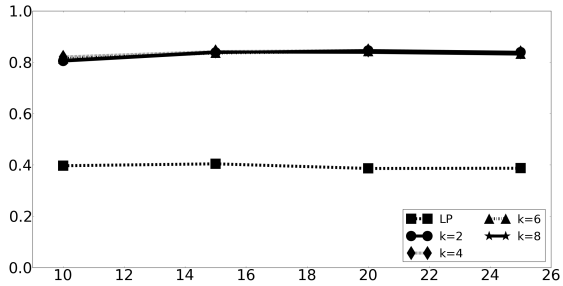

Figure 4: The fraction of cases in which the LP-based algorithm and our algorithm return an equilibrium with independent punishment strategies, taken on average across GAMUT classes with 3-player games, as a function of n. Each point is based on 1000 samples from each game class.

Conclusion

Narrowly viewed, the contribution of this paper is a simple algorithm for finding Nash equilibria of infinitely repeated games that is often much faster than the standard algorithm based on linear programming. However, we believe that there is also a broader interpretation that is perhaps even more significant to AI. This broader interpretation is that, to find equilibria for the kinds of complex, messy, general-sum repeated games that we encounter in the real world, it is generally not necessary to have a complete and perfect model of the game and to compute punishment strategies that are exactly optimal. All that is needed is to find punishment strategies that have a sufficient deterrent effect, and this does not always require difficult optimization problems to be solved (we simply used random samples). This view of equilibrium computation seems to align well with the notion of "game theory pragmatics" (Shoham 2008a; 2008b).

Acknowledgements

We thank ARO and NSF for support under grants W911NF-12-1-0550, W911NF-11-1-0332, IIS-0953756, and CCF-1101659. We would also like to thank our anonymous reviewers for their helpful feedback.

References

Adler, I. (forthcoming). The equivalence of linear programs and zero-sum games. *International Journal of Game Theory*.

Borgs, C.; Chayes, J.; Immorlica, N.; Kalai, A. T.; Mirrokni, V.; and Papadimitriou, C. 2010. The myth of the Folk Theorem. *Games and Economic Behavior* 70(1):34–43.

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

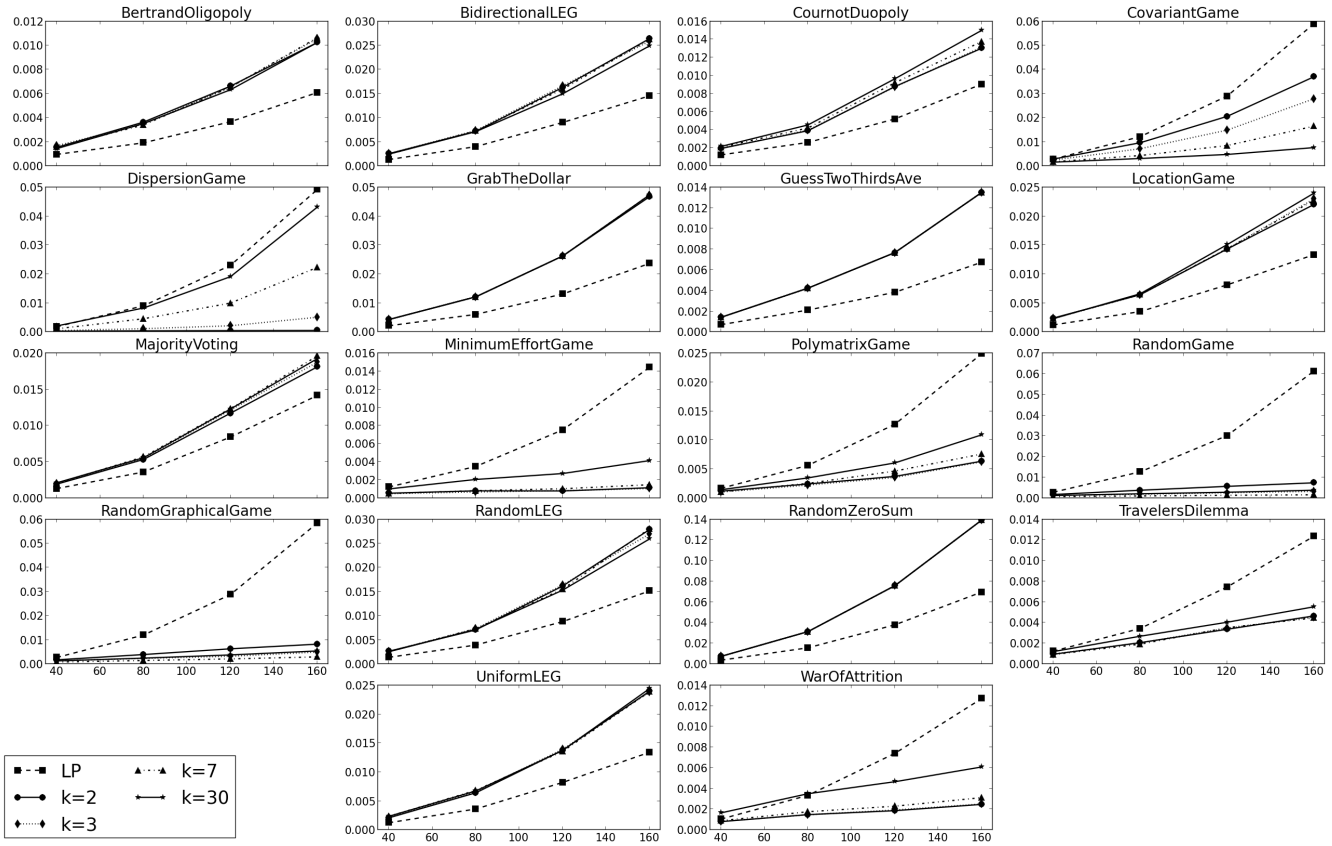

Figure 2: Average runtimes in seconds of the LP-based algorithm and our algorithm for various values of k, for various GAMUT classes with 2-player games, as a function of n. Each point is an average of 1000 samples.

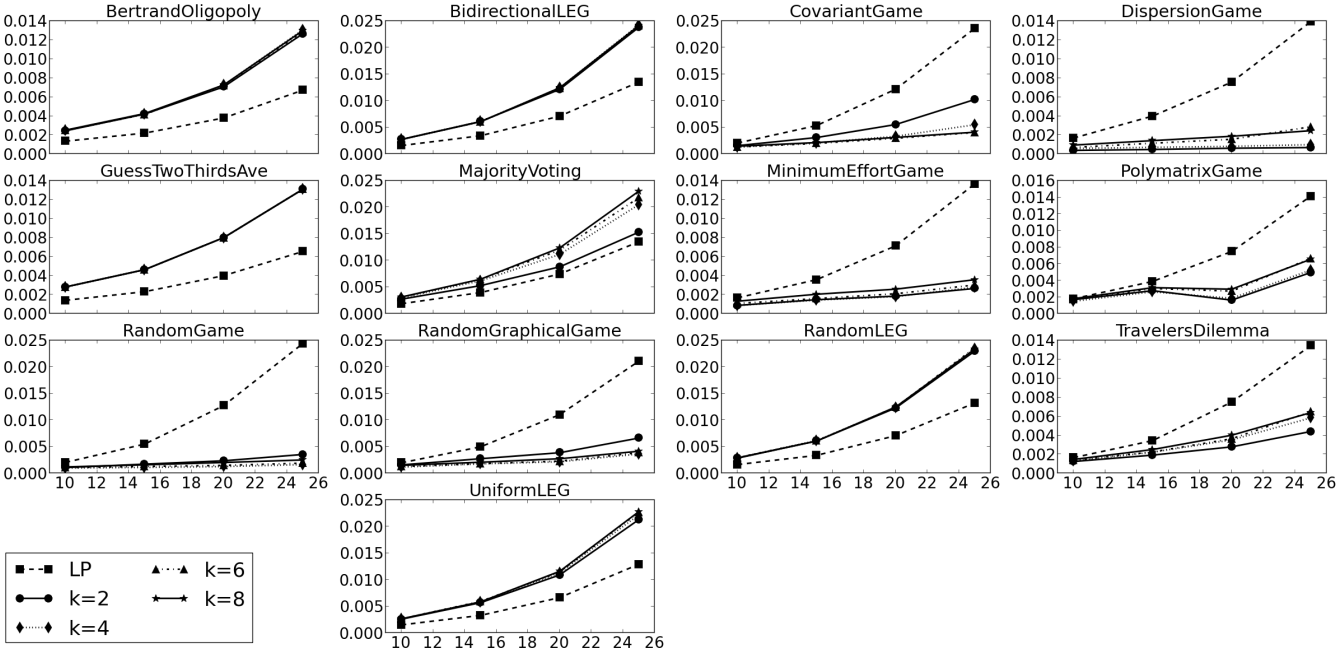

Figure 3: Average runtimes in seconds of the LP-based algorithm and our algorithm for various values of k, for various GAMUT classes with 3-player games, as a function of n. Each point is an average of 1000 samples.

Dantzig, G. 1951. A proof of the equivalence of the programming problem and the game problem. In Koopmans, T., ed., *Activity Analysis of Production and Allocation*. John Wiley & Sons. 330–335.

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

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

Halvorson, E.; Conitzer, V.; and Parr, R. 2009. Multi-step multi-sensor hider-seeker games. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)*, 159–166.

Hansen, K. A.; Hansen, T. D.; Miltersen, P. B.; and Sørensen, T. B. 2008. Approximability and parameterized complexity of minmax values. In *Proceedings of the Fourth Workshop on Internet and Network Economics (WINE)*, 684–695.

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. In *Proceedings of the Tenth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 327–334.

Kontogiannis, S. C., and Spirakis, P. G. 2008. Equilibrium points in fear of correlated threats. In *Proceedings of the Fourth Workshop on Internet and Network Economics (WINE)*, 210–221.

Littman, M. L., and Stone, P. 2005. A polynomial-time Nash equilibrium algorithm for repeated games. *Decision Support Systems* 39:55–66.

Nudelman, E.; Wortman, J.; Leyton-Brown, K.; and Shoham, Y. 2004. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 880–887.

Shoham, Y. 2008a. Computer science and game theory. *Communications of the ACM* 51(8):74–79.

Shoham, Y. 2008b. Game theory pragmatics: A challenge for AI. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1606–1608.