Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence

Quick Polytope Approximation of all Correlated Equilibria in Stochastic Games

Liam MacDermed, Karthik S. Narayan, Charles L. Isbell, Lora Weiss
Georgia Institute of Technology
Robotics and Intelligent Machines Laboratory
Atlanta, Georgia 30332
liam@cc.gatech.edu, karthik.narayan@gatech.edu, isbell@cc.gatech.edu, lora.weiss@gtri.gatech.edu

Abstract

Stochastic or Markov games serve as reasonable models for a variety of domains from biology to computer security, and are appealing due to their versatility. In this paper we address the problem of finding the complete set of correlated equilibria for general-sum stochastic games with perfect information. We present QPACE – an algorithm orders of magnitude more efficient than previous approaches while maintaining a guarantee of convergence and bounded error. Finally, we validate our claims and demonstrate the limits of our algorithm with extensive empirical tests.

1 Introduction and Related Work

Stochastic games naturally extend Markov decision processes (MDPs) in multi-agent reinforcement learning problems. They can model a broad range of interesting problems including oligopoly and monetary policy (Amir 2001), network security and utilization (Nguyen, Alpcan, and Basar 2010), and phenotype-expression patterns in evolving microbes (Wolf and Arkin 2003). Unfortunately, typical applications of stochastic games are currently limited to very small and simple games. Our primary objective in this work is to make it feasible to solve larger and more complex stochastic games.

There have been many approaches to solving stochastic games in game theory and operations research, mostly focused on two-player, zero sum games or repeated games. One line of work uses a recursive "self-generating set" approach that has a very similar flavor to the work we present here (Abreu 1988; Cronshaw 1997; Judd, Yeltekin, and Conklin 2003; Horner et al. 2010). A recent example of this work is that of Burkov and Chaib-draa (2010) where the set of sub-game perfect Nash equilibria is found via a repeatedly finer grain tiling of the set. Unfortunately, none of these approaches has appealing error or computational guarantees able to generalize to stochastic games.

In artificial intelligence, the field of multi-agent learning has produced a particularly promising line of research employing a modified version of Bellman's dynamic programming approach to compute the value function (Greenwald and Hall 2003; Littman 2001; Hu and Wellman 2003). This approach has lead to a few successes, particularly in the zero-sum case; however, value-function based approaches have been proven to be insufficient in the general case (Zinkevich, Greenwald, and Littman 2005). In the wake of this negative result, a new line of research has emerged that replaces the value-function with a multi-dimensional equivalent referred to as the *achievable set*.

Murray and Gordon (2007) derive an intractable algorithm for calculating the exact form of the achievable set based Bellman equation and proved correctness and convergence; however, their approximation algorithm is not generally guaranteed to converge with bounded error, even with infinite time. MacDermed and Isbell (2009) solved these problems by targeting ϵ-equilibria instead of exact equilibria and by approximating the closed convex hull with a bounded number of vertices. Their algorithm bounded the final error introduced by these approximations. While MacDermed and Isbell's algorithm scales linearly with the number of states, it becomes infeasible when there are more than two players with three actions each.

In this paper we adopt a new representation of achievable sets and present a new algorithm, QPACE, that reduces MacDermed and Isbell's approach to a series of efficient linear programs while maintaining their theoretical guarantees. We prove correctness and conduct extensive empirical tests demonstrating QPACE to be several orders of magnitude faster while providing identical results.

2 Background

Stochastic games extend MDPs to multiple agents by using vectors (one element per player) to represent actions and rewards rather than the scalars used in MDPs. In our work, we assume agents are rational and attempt to maximize their long term expected utility with discount factor γ. We assume agents can communicate freely with each other (cheap talk). We also assume that agents observe and remember past joint-actions, allowing agents to change behavior and threaten or punish based on each other's actions (*e.g.* "if you do X, I'll do Y").

We formally represent a stochastic game by the tuple $\langle I, S, s^0, A, P, R\rangle$. The components in the tuple include the finite set of players I, where we let $n = |I|$; the state space S, where each $s \in S$ corresponds to a normal form

game; a set of actions A; a transition probability function $P : S \times A^n \times S \to \mathbb{R}$, describing the probability of transitioning from one state to another given a joint-action $\vec{a} = \langle a_1, \cdots, a_n \rangle$; and a reward function $R : S \times A^n \to \mathbb{R}^n$ describing the reward a state provides to each player after a joint-action. Agents start in initial state s^0. They then repeatedly choose joint-actions, receive rewards, and transition to new states.

When solving stochastic games, we target *correlated equilibria* (CE), which generalize Nash equilibria. A CE takes the form of a probability distribution across joint-actions. Agents observe a shared random variable corresponding to joint-actions drawn from the CE's distribution. This informs agents of their prescribed action without directly revealing other agents' actions; however, information about other agents' actions is revealed in the form of a posterior probability. The distribution is a CE when no agent has an incentive to deviate. CEs only require a shared random variable, but equilibrium selection without communication is an open problem and beyond the scope of this paper; hence our assumption of cheap talk.

2.1 Achievable Sets

The core idea of our algorithm involves an *achievable set function* $V : S \to \{\mathbb{R}^n\}$, the multi-agent analogy to the single agent value-function. As a group of n agents follow a joint-policy, each player $i \in I$ receives rewards. The discounted sum of these rewards is that player's *utility*, u_i. The vector $\vec{u} \in \mathbb{R}^n$ containing these utilities is known as the *joint-utility*, or value-vector. Thus, a joint-policy yields a joint-utility in $\mathbb{R}^n$. If we examine all mixed joint-policies starting from state s, discard those not in equilibrium, and compute all the joint-utilities of the remaining policies we will have a set of points in $\mathbb{R}^n$: the *achievable set*.

An achievable set contains all possible joint-utilities that players can receive using policies in equilibrium. Each dimension represents the utility for a player (Figure 1). This definition is valid for any equilibrium solution concept, but QPACE can only compute achievable sets representable as convex polytopes, such as those from CE. Since this set contains all attainable joint-utilities, it will contain the optimal joint-utility for *any* definition of "optimal." From this achievable set, an optimal joint-utility can be chosen using a bargaining solution (Myerson 1991). Once the agents in the game agree on a joint-utility, a policy can be constructed in a greedy manner for each player that achieves the targeted utilities (Murray and Gordon 2007), similar to the procedure in which a value function can be used to construct a policy in single-agent reinforcement learning. As agents care only about the utility they achieve and not about the specific policy they use, computing the achievable set for each state solves the stochastic game.

2.2 Augmented Stage Games and Threats

When choosing which action to execute, agents consider not only their immediate reward but also the long term utility of the next state in which they may arrive. The expected joint-utility for each joint-action is known as the *continuation utility*, $\overrightarrow{cu_{\vec{a}}}$. Because the agents consider both their immediate

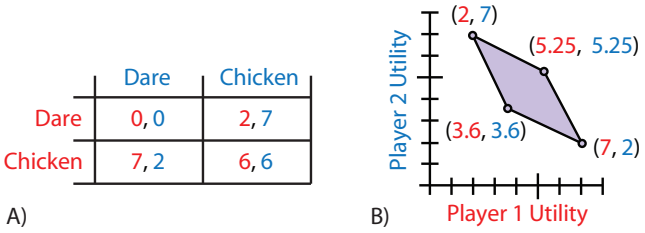

Figure 1: A) The game of chicken. B) The achievable set of correlated equilibria for chicken. For each joint-utility within the region there exists a correlated equilibrium yielding the given rewards for each player. For example the point (5.25, 5.25) is achieved by the agents playing (chicken, chicken) with probability $\frac{1}{2}$ while playing (dare, chicken), and (chicken, dare) each with probability $\frac{1}{4}$.

reward and their continuation utility they are in essence playing an *augmented stage game* with payoffs $R(s, \vec{a}) + \overrightarrow{cu_{\vec{a}}}$.

Assuming that the players cooperate with each other, they choose to jointly play any equilibrium policy in the successor state s'. Thus, the continuation utility $\overrightarrow{cu_{\vec{a}s'}}$ of each successor state may be any point in the achievable set $V(s')$ of that state. We call the set of possible continuation utilities, $Q(s, \vec{a})$, the *action achievable set function*. When there is only one successor, i.e. $P(s'|s, \vec{a}) = 1$, then $Q(s, \vec{a}) = V(s')$. When there is more than one successor state, the continuation utility is the expectation over these utilities, i.e.

$$Q^*(s, \vec{a}) = \left\{ \sum_{s'} P(s'|s, \vec{a}) \overrightarrow{cu_{\vec{a}s'}} \mid \overrightarrow{cu_{\vec{a}s'}} \in V^*(s') \right\} \quad (1)$$

It is advantageous for players to punish other players who deviate from the agreed upon joint-policy. Players accomplish this by changing their continuation utility in the event of defection. The harshest threat possible (which maximizes the achievable set) is a global *grim trigger strategy*, where all other players cooperate in choosing policies minimizing the defecting player's rewards for the rest of the game. The grim trigger *threat-point* ($\overrightarrow{gt_{\vec{a}i}}$) is a continuation utility that players choose for each joint-action $\vec{a}$ in the event that player i defects. This utility can be calculated as a zero-sum game using Nash-Q learning (Hu and Wellman 2003) independently at initialization. While grim trigger threats can lead to subgame imperfect equilibria, extending our algorithm to compute subgame perfect equilibria is easy (see Section 5).

3 The QPACE Algorithm

Achievable set based approaches replace the value-function, $V(s)$, in Bellman's dynamic program with an achievable set function – a mapping from state to achievable set. We update the achievable set function over a series of iterations, akin to value iteration. Each iteration produces an improved estimate of the achievable set function using the previous

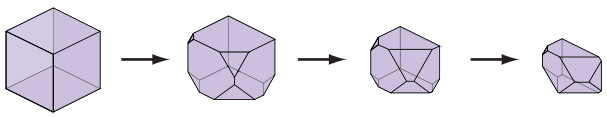

Figure 2: An example achievable set contraction

iteration's estimate. The achievable set function is initialized to some large over-estimate and the algorithm shrinks the sets during each iteration until insufficient progress is made, at which point the current estimate of the achievable set function is returned (Figure 2).

The general outline of QPACE per iteration is:

1. Calculate the action achievable sets $Q(s, \vec{a})$, giving us the set of possible continuation utilities.
2. Construct a set of inequalities that defines the set of correlated equilibria.
3. Approximately project this feasible set into value-vector space by solving a linear program for each hyperplane of our achievable set polytope $V(s)$.

3.1 Representing Achievable Sets

QPACE represents each achievable set as a set of m linear inequalities with fixed coefficients. Recall that the achievable set is closed and convex, and can be thought of as an n-dimensional polytope (one dimension for each player's utility). Polytopes can be defined as an intersection of half-spaces. Each of the m half-spaces $j \in H$ consists of a normal H_j and an offset b_j such that the achievable joint-utility x is restricted by the linear inequality $H_j x \leq b_j$. The half-space normals form a matrix $H = [H_1, \ldots, H_m]$, and the offsets a vector $b = \langle b_1, \ldots, b_m \rangle$. For example, the polytope depicted in Figure 3c can be represented by the equation $Hx \leq b$ where H and b are as shown.

While any achievable set may be represented by an intersection of half-spaces, it may require an unbounded number. Such a growth in the complexity of achievable sets does indeed occur in practice; therefore, we compute an approximation using half-spaces sharing the same fixed set of normals. Each of our polytopes differ only in their offsets, so QPACE must only store the b vector for each $V(s)$ and $Q(s, \vec{a})$. The normals H are chosen at initialization to enable a regular approximation of a Euclidean ball (Figure 3a).

To approximate an achievable set, we find the minimum offsets such that our approximation completely contains the original set; that is, we retain each hyperplane's normal but contract each offset until it barely touches the given polytope. This process is shown for the game of chicken in Figure 3. We call this the *regular polytope approximation (RPA)* of polytope P using normals H where $RPA(H, P)_j = \max_{x \in P} [H_j \cdot x]$ for each offset j. We say that H *permits error* ϵ if $\|RPA(H, P) - P\| \leq \epsilon$. RPA was first presented in MacDermed and Isbell (2009).

Previous achievable set based methods have represented their achievable sets in a number of different ways: convex combinations of vertices (Mac Dermed and Isbell 2009; Gordon 2007), polytopes of arbitrary complexity (Judd, Yeltekin, and Conklin 2003), and as unions of hypercubes (Burkov and Chaib-draa 2010). QPACE's representation allows for quicker computation of each key step in MacDermed and Isbell's (2009) approach leading to an algorithm several orders of magnitude faster than previous algorithms.

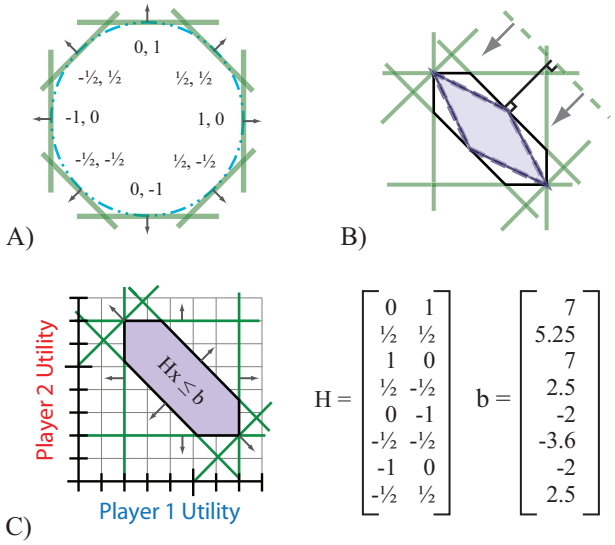

Figure 3: The QPACE representation. A) A Euclidean ball surrounded by half-spaces suitable for a regular polytope approximation (RPA). The normals of each half-space are illustrated. Note that the half-spaces approximate the inscribed dotted circle. B) The achievable set from Figure 1b being approximated using the RPA from (A). Each half-space of the Euclidean ball is moved such that it touches the achievable set at one point. C) The resulting polytope approximation whose normals and offsets are specified in H and b.

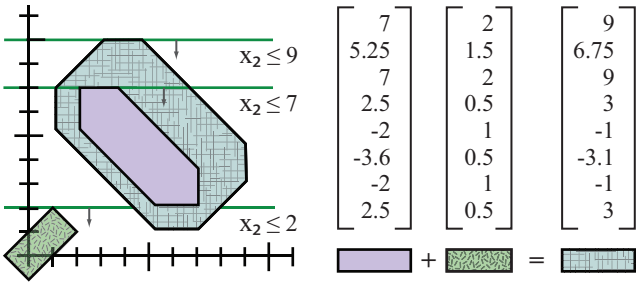

Figure 4: An example approximate Minkowski sum of RPA polytopes. Our method is exact in two dimensions and permits only ϵ error in higher dimensions.

3.2 Computing Action Achievable Sets

At the beginning of each iteration, QPACE calculates the action-achievable set $Q(s, \vec{a})$ for each state s and each joint action $\vec{a}$ in the stochastic game. We do so by rewriting Equation 1 using Minkowski addition. Given two sets A and B, their Minkowski sum $A + B$ is defined as $A + B = \{a+b \mid a \in A, b \in B\}$. Efficient computation of Minkowski addition of two convex polytopes of arbitrary dimension d is an open and active problem, with complexity $O(|A|^d)$ where $|A|$ is the number of half-spaces; however we can approximate a Minkowski sum efficiently using our representation as the sum of the offsets (Figure 4). We prove this result:

Lemma 3.0.1. *Suppose that A and B represent polytopes ($Hx \leq b^A$) and ($Hx \leq b^B$) and H permits only ϵ error as discussed in Section 3.1. Then the weighted Minkowski sum $\alpha A + \beta B$ for constants $\alpha, \beta \in \mathbb{R}^+$ and $\alpha + \beta = 1$ is*

approximated by the polytope $Hx \leq \alpha \cdot b^A + \beta \cdot b^B$ with at most ϵ relative error.

Proof. Noting $RPA(H, A) \leq \epsilon$ and $RPA(H, B) \leq \epsilon$,

$$\begin{aligned} RPA(H, \alpha A + \beta B) &= \max_{x \in A} \max_{y \in B} [\alpha(H_i \cdot x) + \beta(H_j \cdot y)] \\ &= \alpha \cdot \max_{x \in A} [H_j \cdot x] + \beta \cdot \max_{y \in B} [H_j \cdot y] \\ &= \alpha \cdot RPA(H, A) + \beta \cdot RPA(H, B) \\ &\leq \epsilon \cdot (\alpha + \beta) \leq \epsilon, \end{aligned}$$ □

Note that multiple additions do not increase relative error. Equation 1 rewriten as a weighted Minkowski sum becomes: $Q(s, \vec{a}) = \sum_{s'} P(s'|s, \vec{a}) V(s)$. To improve the performance of subsequent steps we scale the set of continuation utilities by the discount factor γ and add the immediate reward (translating the polytope). The final offsets for the action-achievable sets may be computed for each $j \in H$ as:

$$\boxed{Q(s, \vec{a})_j = R(s, \vec{a}) \cdot H_j + \gamma \sum\nolimits_{s'} P(s'|s, \vec{a}) V(s')_j} \quad (2)$$

3.3 Defining the Set of Correlated Equilibria

Calculating the set of CE in a normal form stage game (*e.g.* Figure 1b) is straightforward. A probability distribution X over joint-actions (with $x_{\vec{a}}$ being the probability of choosing joint-action $\vec{a}$) is in equilibrium if and only if the reward of following the prescribed action is no worse than taking any other action. More formally, X is a CE if and only if in state s, for each player i, for distinct actions α, $\beta \in A_i$, where $\vec{a}^{(\alpha)}$ means joint-action $\vec{a}$ with player i taking action α:

$$\sum\nolimits_{\vec{a}} x_{\vec{a}^{(\alpha)}} R(s, \vec{a}^{(\alpha)})_i \;\geq\; \sum\nolimits_{\vec{a}} x_{\vec{a}^{(\alpha)}} R(s, \vec{a}^{(\beta)})_i \quad (3)$$

These rationality constraints are linear inequalities and together with the probability constraints $\sum x_{\vec{a}} = 1$ and $x_{\vec{a}} \geq 0$ define a polytope in $\mathbb{R}^n$. Any point in this polytope represents a CE which yields a value-vector $\sum_{\vec{a}} x_{\vec{a}} R(s, \vec{a})$. The union of all such value-vectors (the polytope projected into value-vector space) is the achievable set of state s when agents do not consider utilities gained from future states.

Agents do not play a single normal form game. Instead they play the augmented game where in order for an agent to make a decision in the current state, they must know which utility among the many possible in $Q(s, \vec{a})$ they will receive in each successor state. Therefore, a CE of the augmented stage game consists of both a probability $x_{\vec{a}}$ for each joint-action and an expected continuation utility $\overrightarrow{cu_{\vec{a}i}}$ for each player and joint-action, resulting in $(n+1)|A|^n$ variables. Given a state s, the set of possible CEs over the variables $x_{\vec{a}}$ and $\overrightarrow{cu_{\vec{a}i}}$ of the augmented game becomes:

$$\text{For each player } i, \text{ distinct actions } \alpha, \beta \in A_i,$$
$$\sum_{\vec{a} \in A^n} x_{\vec{a}^{(\alpha)}} \overrightarrow{cu_{\vec{a}^{(\alpha)}i}} \geq \sum_{\vec{a} \in A^n} x_{\vec{a}^{(\alpha)}} [\overrightarrow{gt_{\vec{a}^{(\beta)}i}} + R(s, \vec{a}^{(\beta)})_i] \quad (4)$$
$$\text{For each joint-action } \vec{a} \in A^n, \text{ and } j \in H\text{:}$$
$$\overrightarrow{cu_{\vec{a}}} \in Q(s, \vec{a}) \quad (\textit{i.e.} \;\; H_j \cdot \overrightarrow{cu_{\vec{a}}} \leq Q(s, \vec{a})_j \;) \quad (5)$$

The rationality constraints (4) are quadratic because we have a variable for the continuation utility ($\overrightarrow{cu_{\vec{a}i}}$) which must be scaled by our variable for the probability of that joint-action occurring ($x_{\vec{a}}$). We can eliminate this multiplication by scaling the action-achievable set $Q(s, \vec{a})$ in inequality 5 by $x_{\vec{a}}$, making the value of $\overrightarrow{cu_{\vec{a}i}}$ already include the multiplication by $x_{\vec{a}}$. This gives us our polytope in $\mathbb{R}^{(n+1)|A|^n}$ over variables $\overrightarrow{cu_{\vec{a}i}}$ and $x_{\vec{a}}$ of feasible correlated equilibria:

$$\boxed{\begin{array}{c} \text{For each player } i, \text{ distinct actions } \alpha, \beta \in A_i, \\ \sum_{\vec{a} \in A^n} \overrightarrow{cu_{\vec{a}^{(\alpha)}i}} \geq \sum_{\vec{a} \in A^n} x_{\vec{a}^{(\alpha)}} [\overrightarrow{gt_{\vec{a}^{(\beta)}i}} + R(s, \vec{a}^{(\beta)})_i] \\ \sum\nolimits_{\vec{a} \in A^n} x_{\vec{a}} = 1 \text{ and } \forall \vec{a} \in A^n, \; x_{\vec{a}} \geq 0 \\ \text{For each joint-action } \vec{a} \in A^n, \\ \overrightarrow{cu_{\vec{a}}} \in x_{\vec{a}} \, Q(s, \vec{a}) \quad (\textit{i.e.} \;\; H_j \, \overrightarrow{cu_{\vec{a}}} \leq x_{\vec{a}} \, Q(s, \vec{a})_j \;) \end{array}} \quad (6)$$

3.4 Computing the Achievable Set Function

The polytope defined above in (6) is the set of CE in joint-action-probability ($x_{\vec{a}}$) and continuation-utility ($\overrightarrow{cu_{\vec{a}i}}$) space; however, we do not ultimately want the set of CEs only the set of resulting value-vectors $V(s)$. The value that a particular correlated equilibrium presents to a player i is $\sum_{\vec{a}} \overrightarrow{cu_{\vec{a}i}}$. Recall that we are approximating our achievable sets with a fixed set of half-spaces, where each half-space touches a vertex of the exact polytope (Figure 3b). This vertex will correspond to the CE with value-vector that maximizes a weighted average of the agents' utilities, weighted by the half-space's normal. We can find this CE using a linear program where the feasible set is as defined in (6) and the objective function is a dot product of the value to each player and the half-space's normal. The optimal weighted average found will equal the offset for that half-space in the approximating polytope. In every state s, we compute one linear program for each approximating half-space and update its offset to the optimal value found. For each half-space j with normal H_j we compute offset $V(s)_j$ as the maximum objective function value to the following LP:

$$\boxed{V(s)_j = \begin{cases} \max & \sum_{\vec{a}} \sum_{i \in I} H_{j,i} \cdot \overrightarrow{cu_{\vec{a}i}} \\ \text{subject to} & \text{inequalities in (6)} \end{cases}} \quad (7)$$

3.5 Linear Program Solution Caching

Every iteration, QPACE solves one LP (equation 7) for each half-space in every state. Depending on the value of ϵ, the number of half-spaces and hence the number of LPs can become very large. Solving these LPs is the main bottleneck in QPACE. Fortunately, we can dramatically improve the performance of these LPs by taking advantage of the fact that the solutions of many of the LPs do not tend to "change" significantly from iteration to iteration. More precisely, the maximum offset difference for a given achievable set in consecutive iterations is small. For a fixed state, we can therefore expect the solution corresponding to the LP of one iteration to be quite close to that of the previous iteration. LPs typically spend the bulk of their running time

Algorithm 1 The QPACE Algorithm.

Inputs: stochastic game G
half-spaces H permitting ϵ error
Output: achievable sets $\{V\}$

1: **for all** $s \in S, j \in H$ **do**
2: $V(s)_j \Leftarrow \max_{s',\vec{a}}(R(s',\vec{a}) \cdot H_j)/\gamma$
3: **repeat**
4: **for all** $s, \vec{a}, j \in H$ **do**
5: $Q(s,\vec{a})_j =$ equation (2)
6: **for all** $s, j \in H$ **do**
7: $V(s)_j \Leftarrow$ LP (7) starting at BFS-cache(s,j)
8: BFS-cache$(s,j) \Leftarrow$ optimal BFS of the LP
9: **until** $\forall s : V(s)_j$ changed less than $\epsilon/2$
10: **return** $\{V\}$

searching over a space of *basic feasible solutions* (BFS), so choosing an initial BFS close to the optimal one dramatically improves performance. QPACE caches the optimal BFS of each state/half-space calculation and uses the solution as the starting BFS in the next iteration. Empirical tests confirm that after the first few iterations of QPACE the LPs will find an optimal BFS very near the cached BFS (see Figure 5f). In fact, a majority of cached BFSs are already optimal for the new LP. Note that even if the optimal BFS from one iteration is exactly the optimal BFS of the next, the optimal values may be different as the offsets change.

This optimization only applies if a BFS from one iteration, x^*, is a valid BFS in the next. Every LP has identical variables, objectives, and constraints, with subsequent iterations differing only in the offsets b; therefore, x^* is a basic solution for the new LP. While x^* is basic, it may not be feasible for the new LP. However, LP sensitivity analysis guarantees that an optimal BFS from one iteration is a BFS of the dual problem in the next iteration when only b changes. Therefore, we alternate each iteration between solving the primal and dual problems.

3.6 Correctness

In the QPACE algorithm (Algorithm 1) each step except for line 5 performs an equivalent operation to MacDermed and Isbell's (2009), albeit using a different representation. The Minkowski sums in line 5 can now introduce ϵ error into the action achievable sets, doubling the potential error induced at each iteration. To mitigate this factor we run QPACE with twice the precision when there are three or more players. Convergence and error bounds are therefore the same as in Mac Dermed and Isbell (2009). Interestingly even with three or more players our empirical tests show that LP (7) with exact Minkowski sums and LP (7) with our approximate Minkowski sums give identical results.

4 Empirical Results

We ran tests to determine the scalability of QPACE across a number of different dimensions: precision (Figure 5a), number of states (Figure 5b), number of joint-actions (Figure 5c), and number of players (Figure 5d). We ran the algorithms over generated games, random in the state transition and reward functions. Unless otherwise indicated, the games were run with 10 states, 2 players, 2 actions each, and with $\epsilon = 0.05$ and $\gamma = 0.9$. We used the GLPK library as our linear programming solver, and used the FFQ-Learning algorithm (Littman 2001) to compute grim trigger threats.

As users specify more precision, the number of hyperplanes in our approximate Euclidean ball increases exponentially. So, wall clock time increases exponentially as ϵ decreases as Figures 5a confirms. The QPACE algorithm begins to do worse than MacDermed and Isbell (MI09) for very low values of ϵ, especially in QPACE without caching (Figure 5a). To see why, note that if an achievable set can be represented by only a few verticies or half-spaces (*e.g.* a hypercube) than MI09 will only use those verticies necessary while QPACE will always use all half-spaces in H.

Both MI09 and QPACE scale linearly with the number of states independent of other parameters (Figure 5b), however QPACE is about 240 times faster than MI09 with the default parameters. The number of variables used in each LP is linear with respect to the number of joint-actions. LP's in turn are polynomial with respect to the number of variables. Therefore as expected, both algorithms are polynomial with respect to joint-actions (Figure 5c). However, QPACE uses significantly fewer variables per joint-action and therefore has a lower rate of growth.

QPACE scales much better than MI09 with respect to the number of players. The number of joint-actions grows exponentially with the number of players, so both algorithms have exponential growth in the number of players; however, the number of vertices needed to represent a polytope also grows exponentially while the number of half-spaces only grows polynomially. Thus, QPACE is able to handle many more players than MI09.

A natural question to ask is how much each of QPACE's new aspects contribute to the overall efficiency. The change of representation permits three key improvements: fewer variables in the linear programs, an elimination of the need to calculate vertices of the polytopes, fast Minkowski addition, and LP solution caching. By comparing QPACE without caching to MI09 in Figures 5a-c we observe the impact of the first two improvements. We now examine Figures 5e&f to determine the contribution of the last two improvements.

Both QPACE and MI09 employ Minkowski sums when computing the expected continuation utility over future states (when transitions are deterministic, neither algorithm computes Minkowski sums). As the number of possible successor states increases, both algorithms perform more Minkowski additions. Figure 5e graphs the effect of additional successor states for both MI09 and QPACE and shows that while MI09 suffers from a 10% increases in time when the number of successors is large, QPACE is unaffected.

From Figure 5f, we observe that as the number of iterations progresses, the average number of LP iterations involved decreases quickly for QPACE with caching. The algorithm initially takes around 100 LP iterations, and drops to less than 3 LP iterations by the sixth iteration. On the other hand, QPACE without caching starts at around 20 LP

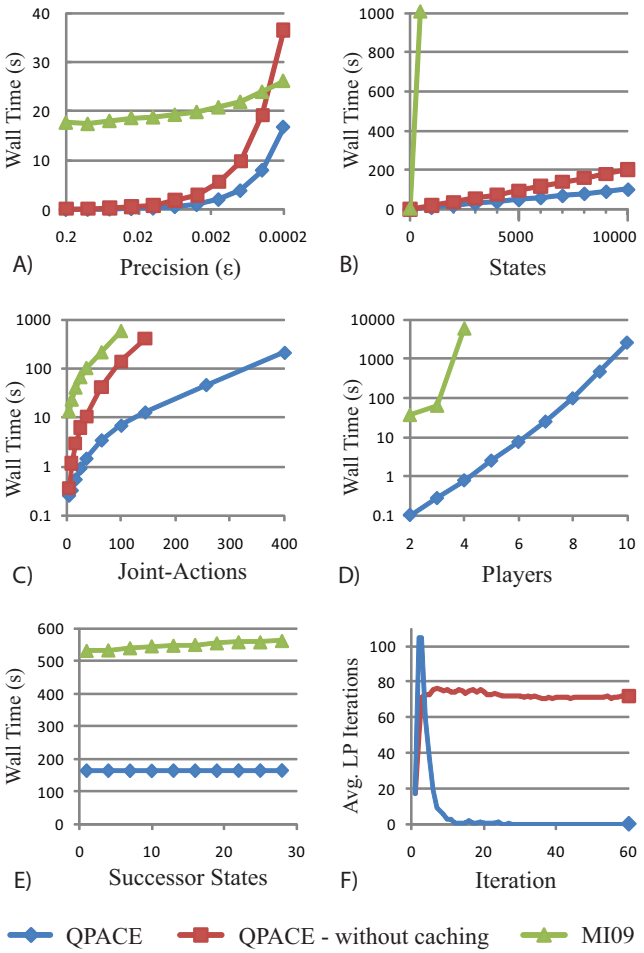

Figure 5: Performance evaluation. A-E) An illustration of how our algorithm scales along a number of variables. Note that some axis are logarithmic. F) Average number of iterations each LP runs for with and without caching over iterations of QPACE.

iterations and plateaus to a consistent 70 LP iterations. The graphs demonstrate that LP caching does in fact make a significant difference in running time. After the tenth iteration, caching consistently reduces the time per iteration from 1.5 seconds down to 0.2 seconds. In the long run, LP caching contributes an order of magnitude speed boost.

5 Conclusion and Extensions

QPACE finds all correlated equilibria of a stochastic game orders of magnitude more efficiently than previous approaches, while maintaining a guarantee of convergence. Further, no returned equilibria provides more than ϵ incentive to deviate. QPACE's regular half-space representation and LP caching scheme allows QPACE to solve larger and more complex games.

QPACE can easily be modified to produce sub-game perfect equilibria by making the threat points $\overrightarrow{gt_{\vec{a}i}}$ variables in (6) and constraining them in a similar way to the $\overrightarrow{cu_{\vec{a}i}}$; we then employ the same trick described in section 3.3 to remove the resulting quadratic constraints. QPACE can also be extended to work in games of imperfect monitoring and imperfect recall. In these games player's don't observe actions so they can't use threats and can't choose different continuation points for each joint-action. Thus $\overrightarrow{gt_{\vec{a}i}} = \overrightarrow{cu_{\vec{a}i}}$ and the values of $Q(s, \vec{a})_j$ are interdependent across actions, forcing equations (1) and (6) to become one large set of inequalities.

The empirical results suggest that a good estimate requires many fewer half-spaces than a full Euclidean ball. A straight forward extension would be to start with only a few half-spaces and dynamically add half-spaces as needed.

References

Abreu, D. 1988. On the theory of infinitely repeated games with discounting. *Econometrica* 56(2):383–96.

Amir, R. 2001. Stochastic games in economics and related fields: an overview. CORE Discussion Papers 2001060.

Burkov, A., and Chaib-draa, B. 2010. An approximate subgame-perfect equilibrium computation technique for repeated games. *CoRR* abs/1002.1718.

Cronshaw, M. B. 1997. Algorithms for finding repeated game equilibria. *Computational Economics* 10(2):139–68.

Gordon, G. J. 2007. Agendas for multi-agent learning. *Artificial Intelligence* 171(7):392 – 401.

Greenwald, A., and Hall, K. 2003. Correlated-q learning. In *Proc. 20th International Conference on Machine Learning (ICML)*, 242–249.

Horner, J.; Sugaya, T.; Takahashi, S.; and Vieille, N. 2010. Recursivemethods in discounted stochastic games: an algorithm for $\delta \to 1$ and a folk theorem. *Econometrica*.

Hu, J., and Wellman, M. 2003. Nash q-learning for general-sum stochastic games. In *Journal of Machine Learning Research 4:1039-1069.*

Judd, K. L.; Yeltekin, S.; and Conklin, J. 2003. Computing supergame equilibria. *Econometrica* 71(4):1239–1254.

Littman, M. L. 2001. Friend-or-foe Q-learning in general-sum games. In *Proc. 18th International Conf. on Machine Learning (ICML)*, 322–328.

Mac Dermed, L., and Isbell, C. 2009. Solving stochastic games. In *Advances in Neural Information Processing Systems 22*. 1186–1194.

Murray, C., and Gordon, G. J. 2007. Multi-robot negotiation: Approximating the set of subgame perfect equilibria in general-sum stochastic games. In *Advances in Neural Information Processing Systems 19*. 1001–1008.

Myerson, R. B. 1991. *Game Theory: Analysis of Conflict.* Harvard University Press, Cambridge.

Nguyen, K. C.; Alpcan, T.; and Basar, T. 2010. Stochastic games for security in networks with interdependent nodes. *CoRR* abs/1003.2440.

Wolf, D. M., and Arkin, A. P. 2003. Motifs, modules and games in bacteria. *Current Opinion in Microbiology* 6:125134.

Zinkevich, M.; Greenwald, A.; and Littman, M. L. 2005. Cyclic equilibria in markov games. In *Proceedings of Neural Information Processing Systems*.