Computing Supergame Equilibria

Kenneth L. Judd
Hoover Institution
Stanford, CA 94305
judd@hoover.stanford.edu

Sevin Yeltekin
KGSM-MEDS
Northwestern University
2001 Sheridan Rd.
Evanston, IL 60208
s-yeltekin@kellogg.nwu.edu

James Conklin
Lehman Brothers
New York, New York

This revision: September, 2000*

Abstract. We present a general method for computing the set of supergame equilibria in infinitely repeated games with perfect monitoring and public randomization. We present a three-step algorithm which constructs a convex set containing the set of equilibrium values, constructs another convex set contained in the set of equilibrium values, and produces strategies which support equilibrium values. We explore the properties of this algorithm by applying it to familiar games. We find that the algorithm produces high quality approximations at moderate computational cost.

Keywords: Computational methods, repeated games, Nash equilibrium

*This work was supported by NSF grant SES-9012128, SES-9309613, and SES-9708991. This paper is an extension and final version of Conklin and Judd (1993). We gratefully acknowledge comments of seminar participants at Northwestern University and SITE.

1. Introduction

The nature of repeated interaction has been extensively studied in the repeated game literature. The theory of repeated games has produced important qualitative results, but it is difficult to apply the theory quantitatively. This paper presents methods for computing the set of subgame perfect equilibria in infinitely repeated games with perfect monitoring. Our approach is recursive, following the "self-generating set" constructions of Abreu (1988), Abreu, Pearce and Stacchetti (1986, 1990), and Cronshaw and Luenberger (1994). Our algorithm can quantitatively investigate many of the properties of the set of all Nash equilibrium payoffs in infinitely repeated games. The method presented in this paper is general, being applicable to a large variety of games studied in industrial organization, contract theory, and dynamic policy analysis. Furthermore, the method can be extended in many directions such as infinite-horizon games with state variables.

The recursive theory of supergames introduces the concept of self-generating sets of payoffs to characterize the set of equilibria. The set of equilibrium values may not be well-behaved. For example, Sorin (1986) studies an infinitely repeated Prisoner's Dilemma game and shows that its equilibrium value set is a square plus two line segments. In particular, the equilibrium value set is neither convex nor star-shaped, and does not equal the closure of its interior. One suspects that equilibrium values sets are often difficult to approximate in a finitistic, computable fashion in any reliable way. To make the problem tractable, we allow public randomization, as in Cronshaw and Luenberger (1994). This is advantageous since the resulting set of equilibrium values is convex and convex sets can be approximated in several ways. The numerical methods presented use efficient ways to approximate the set-to-set mappings used in APS and Cronshaw and Luenberger and their fixed points, and also compute actions and strategies which support equilibrium values.

Any convex set is characterized by its boundary. We can therefore approximate a convex set by approximating its boundary. We choose piecewise linear methods for this approximation, and argue that they are the best possible approximations given the nature of repeated games and our objectives. We develop two methods that can produce "lower bounds" to the true solution, that is, convex sets that are contained in the true solution; we call these *inner approximations*. We also present an efficient method for constructing *outer approximations*, that is, supersets of the true set of solutions. The ability to

produce both inner and outer approximations is valuable since these approximations will allow us to prove that some values can be achieved in equilibrium and prove that some other values cannot be achieved in equilibrium. There will always be some values for which we cannot determine whether they are equilibrium values, but our inner and outer approximations typically leave only small amounts of ambiguity. We also want to know the actions and strategies that support equilibrium values. To do this, we use *ray methods.* The final algorithm consists of three steps which accomplish three tasks: construct an outer approximation of the equilibrium value set, construct an inner approximation of the equilibrium value set, and use a ray method applied to an inner approximation to construct strategies. The complete algorithm produces much of the information we want about equilibria in a supergame.

Our algorithm has properties which are unusual for numerical methods. It produces both upper and lower bounds for the set of equilibria. Furthermore, we show that these bounds are tight in many familiar games. This property is much better than usual for computational methods. For example, methods for solving nonlinear equations, such as Newton's method, the Scarf method (1967), and homotopy methods (e.g., Eaves, 1972), will produce an infinite sequence of approximations which converge to a true solution. However, in real computations one needs to stop this sequence at some finite point without knowing how far that point is from the true solution. Few numerical algorithms can even construct a set which definitely contains the true solution. The fact that we often get tight lower and upper bounds on the set of equilibrium values means that our algorithm often produces a proof of whether or not a value can be achieved in equilibrium and does this for nearly all values of interest.

Our algorithm has strong approximation properties because it exploits monotonicity properties of the crucial set-to-set operator, generally denoted $B(W)$, defining self-generating value sets. However, it has those strong properties only because the algorithm is carefully constructed to preserve monotonicity. The methods we use appear to be the best possible methods given the objective of constructing upper and lower bounds. Other methods which don't preserve monotonicity may be faster, but they are unlikely to be reliable since the underlying operator is not necessarily continuous. Our constructions can handle any finite-action, finite-player game without making any continuity assumptions about the dependence of equilibrium value sets on the parameters of the game. Also, the

algorithm is robust in that it does not need good initial guesses, whereas many nonlinear equation algorithms such as Newton and Gaussian solution methods typically need good initial guesses.

There are three basic reasons why we want reliable and fast algorithms. First, the only way to convincingly demonstrate comparative dynamic or comparative static propositions would be to calculate hundreds or thousands of examples. Slow procedures would be of little value in this case, and algorithms will too often fail and make one question all the results. Second, more efficient methods will be able to analyze more complex economic models. Third, we want algorithms which can conceivably be used for econometric estimation of the underlying nonlinear models.

Speed and reliability will be particularly important in extensions of this algorithm to more complex games, such as games with state variables and combinations of strategic and competitive players. Atkeson (1991) extended the APS method to a game with a state variable in the context of international borrowing and lending. APS and Atkeson analyze games with large players, i.e players whose actions affect the choices of other players. Games between a large player and a continuum of small players, mainly in macroeconomic contexts, are studied in Phelan and Stacchetti (1999) and Sleet (1996). They show that such environments can also be analyzed using APS methods. In these dynamic contexts, the APS operator is viewed as a mapping between correspondences, or equivalently, between the graphs of correspondences. In the repeated setting, described in this paper, lotteries are introduced to guarantee the convexity of the payoff sets. In the dynamic settings, lotteries only guarantee that the candidate payoff correspondences are convex-valued. Consequently, a method of approximating convex-valued correspondences is required before the APS approach can be numerically implemented in the dynamic setting. Two such methods that utilize set-valued step functions are provided by Sleet and Yeltekin (2000), with applications to international lending and a bilateral insurance game with storage.

Section 2 describes the theory of supergame equilibria and its extension to include public randomization. Section 3 discusses the methods for approximating convex sets. Section 4 describes desirable features for approximations of the $B(W)$ operator which is the focus of Abreu-Pearce-Stacchetti theory, algorithms for approximating $B(W)$, and algorithms for computing V. Section 5 applies our algorithms to familiar games and

demonstrates the effectiveness of our algorithms and the importance of various details. Section 6 concludes.

2. SUPERGAMES AND CHARACTERIZATION OF EQUILIBRIUM PAYOFFS

We examine an N-player infinitely repeated game constructed from infinite repetition of a static stage game. The actions of player i in the stage game are in $A_i, i = 1, \cdots, N$. Elements of $A \equiv A_1 \times A_2 \times \cdots \times A_N$ represent all possible combination of player actions and are called *action profiles.* Player i's payoff in the stage game will be $\Pi_i : A \to R$. Abusing notation in the standard way, we define $a_{-i} \equiv (a_1, \cdots, a_{i-1}, a_{i+1}, \cdots, a_N)$. The payoff to the best response of player i to his opponents actions, a_{-i}, is

$$\Pi_i^*(a_{-i}) \equiv \max_{a_i \in A_i} \Pi_i(a_i, a_{-i}).$$

We make the following standard assumptions.

Assumption 1: A_i, $i = 1, \cdots, N$, is a finite set.

Assumption 2: The stage game has a pure strategy Nash equilibrium.

These assumptions limit the range of games we examine, but this is a natural beginning for computing supergame equilibria. If we were to allow continuous strategies then some of our approximation methods would break down and we would lose some strong approximation properties. Even if we were to allow continuous strategies, any computation would only be able to examine a finite number of actions. We make Assumption 1 here to avoid issues better discussed in later work[1]. Assumption 2 is standard in the supergame literature.

We construct the supergame S^∞. The action space of S^∞ is $A^\infty \equiv \times_{i=1}^\infty A$. We assume that player i aims to maximize his discounted payoff

$$\frac{1-\delta}{\delta} \sum_{t=1}^{\infty} \delta^t \Pi_i(a_t)$$

where a_t is the action profile taken in period t. Define the minimum and maximum payoffs: $\underline{\Pi}_i \equiv \min_{a \in A} \Pi_i(a)$, $\overline{\Pi}_i \equiv \max_{a \in A} \Pi_i(a)$. The supergame payoffs are contained

[1] Cronshaw (1997) generalized Conklin and Judd (1993) by examining the case of continuous strategies. We will later discuss the reasons why his continuous strategy procedure does not have the strong approximation properties which our algorithm does.

in the compact set $\mathcal{W} = \times_{i=1}^{n}[\underline{\Pi}_i\,,\ \overline{\Pi}_i]$. Let $V^P \subset \mathcal{W}$ denote the set of all equilibrium supergame payoffs.

Constructing subgame perfect equilibrium values sets involve approximating V^P which may be extremely complex. Fortunately, we can take a dynamic programming approach to the problem, pioneered by APS and adapted for complete information games by Cronshaw and Luenberger. The dynamic programming approach is illustrated in a simple example. Suppose that there are two players, each with two actions, and that the stage game payoff matrix is

	C plays 1	C plays 2
B plays 1	$b_{11},\ c_{11}$	$b_{12},\ c_{12}$
B plays 2	$b_{21},\ c_{21}$	$b_{22},\ c_{22}$

where b_{ij} (c_{ij}) is player B's (C's) payoff if player B (C) plays i (j).

The key insight is that each stage of the repeated game can be represented as a static game with payoffs equal to the stage game payoffs augmented by future payoffs. The present value of future payoffs is called the *continuation value*. In a supergame equilibrium the continuation values depend on the current play. Specifically, at any point in an equilibrium the situation is equivalent to a one-shot game where B (C) receives a discounted payoff of u_{ij}^B (u_{ij}^C) in the future if player B plays i and C plays j in the current stage for some u_{ij}^B (u_{ij}^C). Let $\delta^* = 1 - \delta$. Then at any point in time, the repeated game is equivalent to the static game

	C plays 1	C plays 2
B plays 1	$\delta^* b_{11} + \delta u_{11}^B,\ \delta^* c_{11} + \delta u_{11}^C$	$\delta^* b_{12} + \delta u_{12}^B,\ \delta^* c_{12} + \delta u_{12}^C$
B plays 2	$\delta^* b_{21} + \delta u_{21}^B,\ \delta^* c_{21} + \delta u_{21}^C$	$\delta^* b_{22} + \delta u_{22}^B,\ \delta^* c_{22} + \delta u_{22}^C$

Let $G(u)$ denote the game with these augmented payoffs.

Since we are interested only in subgame perfect equilibria, each Nash equilibrium of S^∞ is a sequence of static equilibria to games of the form $G(u)$ where the continuation values u are elements of V^P. Therefore, if $v \in V^P$, then there is some one-shot action profile, a, and some continuation values, $u_{ij} = (u_{ij}^B, u_{ij}^C) \in V,\ i, j = 1, 2$, such that a is a Nash equilibrium of $G(u)$ and v is the corresponding payoff vector. We exploit the self-referential aspect of V^P: if $v \in V^P$, then there are continuation values in V such that v is the payoff of some augmented game constructed from continuation values in V^P.

This feature leads us to the methods used in APS and Cronshaw and Luenberger. The key to finding the subgame perfect equilibrium payoffs of supergames is the construction of *self-generating* sets. Intuitively, a set $W \subseteq R^N$ is self-generating if each value in W can be supported by continuation values which themselves have values in W. The concept of self-generation can be formalized by the construction of a map, $B^P(W)$. Suppose that W is the set of possible payoffs tomorrow. Let $B^P(W)$ denote the set of payoffs possible today using pure strategies and consistent with Nash play in the augmented game $G(u)$ for some $u \in W$. $B^P(W)$ is formally defined by

$$B^P(W) = \bigcup_{(a,w)\in A\times W} \{(1-\delta)\Pi(a) + \delta w \mid \forall i(IC_i),\ i = 1,2\} \tag{1}$$

where

$$IC_i \equiv ((1-\delta)\Pi_i(a) + \delta w_i) - ((1-\delta)\Pi_i^*(a_{-i}) + \delta \underline{w}_i) \geq 0$$

is the incentive compatibility condition for player i, and

$$\underline{w}_i \equiv \inf_{w\in W}\ w_i$$

is player i's minimum possible continuation value in W. A value b is in $B^P(W)$ if there is some action profile, $a \in A$, and continuation payoff, $w \in W$, such that b is the value for players 1 and 2 of playing a today and receiving the continuation value w tomorrow, and, for each i, player i will choose to play a_i because he believes that to do otherwise will yield him the worst possible continuation payoff, $\underline{w}_i$. Cronshaw and Luenberger (1994) show that the largest fixed-point of the mapping $B^P(W)$ is V^P.

While this definition of V is elegant and intuitive, there are important difficulties associated with computing V^P. The main problem is that V^P can be difficult to represent in a computer. To deal with these problems, we follow Cronshaw and Luenberger (1994) and alter the supergame by including public randomization. More precisely, we assume that in each repetition of the game there is a lottery, $\widetilde{w}(a)$, depending on the actions chosen by the players in that period, that will determine which Nash equilibrium will be played in the next period. Intuitively, this is like having a "sunspot" determine the continuation value since the randomization is a public randomization. Since the payoff is separable over time, the set of possible continuation values of the game is the convex hull of the set of values which can be supported by pure strategies. The strategies are still

pure; it is only the continuation value which is randomized. Cronshaw and Luenberger (1994) showed that the critical map for defining self-generating sets of the game with public randomization is

$$B(W) = co\left(B^P(W)\right) \tag{2}$$

Theorem 1 reviews some obvious properties for $B(W)$.

Theorem 1. *The maps B and B^P are monotone; that is, if $W \supseteq W'$ then $B(W) \supseteq B(W')$ and $B^P(W) \supseteq B^P(W')$*

Proof. See Cronshaw and Luenberger. ■

While the main motivation for adding lotteries is to make the computation easier, it does generalize the notion of subgame perfect Nash equilibrium in an appropriate and interesting fashion. Since one aim of this research is to see what is possible in the absence of contracting, it is natural to add this public randomizing device. The addition of lotteries will often be inessential. In particular, if the set of equilibrium values without lotteries, V^P, is convex, then V^P will also be the set of equilibrium values with lotteries, and our algorithm will approximate the set of equilibria without randomization.[2]

We will use $B(\cdot)$ to construct the set of all possible supergame payoffs, V, of the game with lotteries. Our algorithms will focus on constructing V, not V^P, because V is convex. The key properties of V are summarized in the following theorem:

Theorem 2. *V, the set of equilibrium payoff values under public randomization, is convex and satisfies*

$$V = B(V) = \bigcup_{\substack{W \subseteq \mathcal{W} \\ W \subseteq B(W)}} W = \bigcup_{\substack{W \subseteq \mathcal{W} \\ W = B(W)}} W \tag{3}$$

Proof. See Cronshaw and Luenberger. ■

3. Piecewise Linear Approximations of Convex Sets

We need efficient ways to approximate convex sets in the computer since the value sets in our games are convex. We also want to use approximations which give us useful and relatively precise information about the approximation error. For these reasons we

[2] One could also further generalize the analysis to correlated equilibria. We do not pursue that generalization here.

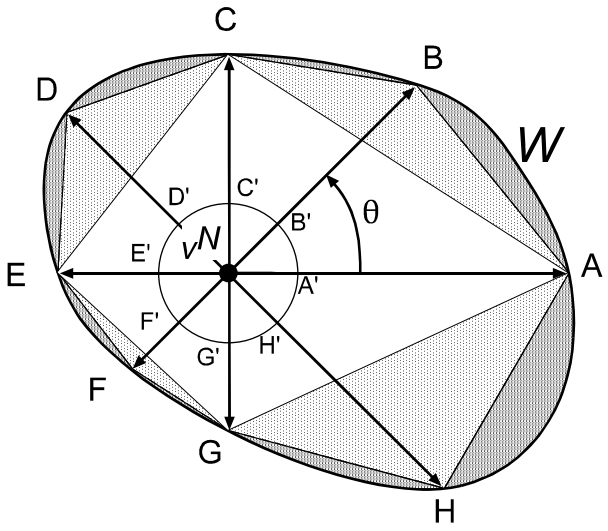


Figure 1: Inner approximations

approximate convex sets with piecewise linear approximations of their boundaries. This approach allows us to use only a finite amount of information to approximate a convex set. Our presentation and figures focus on approximating two-dimensional convex sets, such as W in Figure 1. However, all of the ideas generalize to R^n.

We will use only convex polytopes since they can be represented in a finite fashion and fit in well with other aspects of our algorithms. There are two basic ways to describe convex polytopes. First, if W is a convex polytope, then for some finite set of vertices Z, $W = co(Z)$, the convex hull of Z. We can use Z to represent the convex polytope $co(Z)$. Second, W is the intersection of a finite number of half-spaces and can be represented as a collection of linear inequalities. In this case $W = \cap_{\ell=1}^{L}\{z \mid g_\ell \cdot z \leqq c_\ell\}$ where $g_\ell \in R^N$ is the gradient orthogonal to the face ℓ of W, and $c_\ell \in R$ is a scalar which we shall call a *level* since it is the value of $g_\ell \cdot z$ for any point z on face ℓ. We shall use both ways, vertices and pairs of gradients and levels, of representing convex polytopes.

We refer to two kinds of convex polytope approximations of convex sets W. First, we define an *inner approximation*. Suppose that we are given m points in R^n, $Z = \{P_1, ..., P_m\}$ in a convex set W. For example, let $Z = \{A, B, C, D, E, F, G, H\}$ in Figure 1. The convex hull of Z, $co\,(Z)$, is contained in W and has a piecewise hyperplanar boundary;

it is a convex polytope. In Figure 1, the polygon $ABCDEFGH$ is the boundary of $co(Z)$. Since $co(Z) \subseteq W$, we will call $co(Z)$ the *inner approximation to W generated by Z*. The inner approximation is better as we use more points in W and as they are closer to the boundary of W. For example, the error in approximating W by $ABCDEFGH$ is the dark shaded area in Figure 1. The four-point approximation $ACEG$ has a larger error equal to the dark shaded area plus the light shaded area in Figure 1. The convex hull of Z is also the best possible inner approximation since any convex set which contains Z contains $co\,(Z)$ and $co\,(Z)$ is the largest convex set which is surely in W if all we know is that $Z \subset W$.

Definition 3. *If $Z \subset W \subset R^n$ is a set of m points, then the inner approximation to W generated by Z is $co(Z)$.*

Second, we describe the concept of an *outer approximation*. Suppose we are given m points on the boundary of W, $Z = \{z_1, ..., z_m\}$, and a corresponding set of subgradients, $G = \{g_1, ..., g_m\}$; that is, the hyperplane $z_i \cdot g_i = z \cdot g_i$ is tangent to W at z_i, and the gradients are oriented so that $g_i \cdot w \leqq g_i \cdot z_i$ for $w \in W$. This is illustrated in Figure 2 where we graph the tangent plane and normal for W at $z_i = (x_i, y_i)$ and several other points. The boundary of W has a gradient, $g_i = (s_i, t_i)$, at each $z_i = (x_i, y_i)$. Each tangent hyperplane divides R^n into two pieces; call the half-space containing W the *interior half-space*. The *outer approximation of W generated by* $(Z,\ R)$ is the intersection of the interior half spaces of the supporting hyperplanes. The outer approximation we define is also the most appropriate one since it is the smallest convex set which surely contains W given the information contained in (Z, G). Definition 4 presents this idea formally.

Definition 4. *If Z is a set of m points on the boundary of a convex set $W \subset R^n$ and $G \subset R^n$ a set of m corresponding subgradients oriented such that $\left(z^\ell - w\right) \cdot g^\ell > 0$ for $w \in W$, then the outer approximation of W generated by (Z, G) is*

$$\widehat{W} = \cap_{\ell=1}^{m} \{z \in R^n \mid g_\ell \cdot z \leqq g_\ell \cdot z_\ell\} \tag{4}$$

To compute inner and outer approximations of a set W we need to compute boundary points Z and subgradients G. There are two basic ways to compute inner approximations. The *ray procedure* is illustrated in Figure 1. In R^2, such as in Figure 1, let θ be the polar coordinate angle with origin at v^N. To find an inner approximation of W we choose a

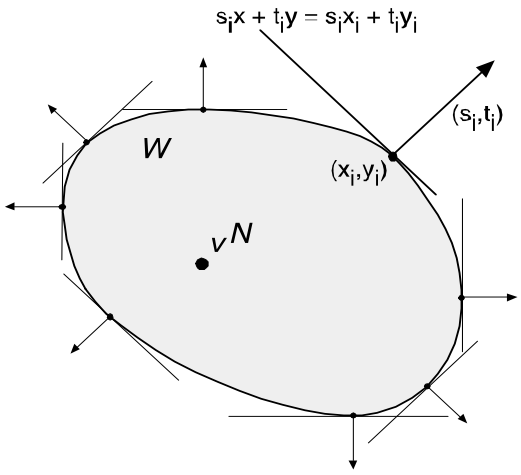


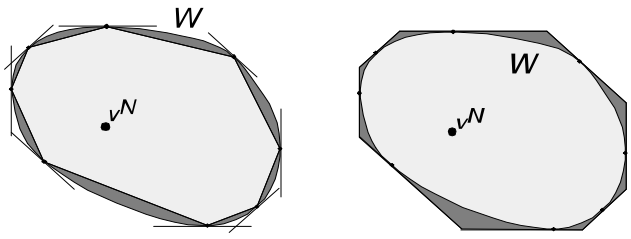
Figure 2: A convex set and supporting hyperplanes

set of θ angles and find where each ray at angle θ intersects the boundary of W. This problem is a linear programming problem if W is a convex polytope. For $W \subset R^n$ more generally, we would choose a set of points on a unit sphere to define rays. For example, in Figure 1, we would specify a circle such as $A'B'C'D'E'F'G'H'$ (which here lies inside of W but that is not necessary) and find where the rays $\overrightarrow{v^N A'}$, $\overrightarrow{v^N B'}$, etc., intersect the boundary of W. In any case, the collection of boundary points is then used to compute an inner approximation.

An *extremal procedure* can be used to compute both an inner and outer approximation. For any subgradient $g \in R^n$, any point z_0 (there may be many such points) which solves the maximization problem

$$\max_{z \in W} \; z \cdot g \tag{5}$$

is a boundary point of $W \subset R^n$ with subgradient g. The problem (5) is a linear programming program if W is a convex polytope, and a convex optimization problem in general. The tangent hyperplane at the solution z_0 is the plane $z \cdot g = z_0 \cdot g$. By repeating this for a variety of subgradients g, we can compute a collection of point-gradient pairs, (Z, G). The resulting collection of boundary points Z can also be used to construct $co\,(Z)$, an inner approximation. Such an inner approximation is illustrated in Figure 3(a). The lightly shaded region is the inner approximation and the darkly shaded area is the approximation



(a) Inner Approx. (b) Outer Approx.

Figure 3:

error. As we add more subgradients, the inner approximation grows to equal the set W.

We can also use (Z, G) in (4) to define an outer approximation. Figure 3(b) displays such an outer approximation for the convex set in Figure 2 where $G = \{-1, 0, 1\}^2$. The outer approximation is the dark plus light shaded region, and the approximation error is the dark shaded area. The error is reduced and the outer approximation converges to W as we use more subgradients.

Computing the convex hull of a set of points efficiently is not an easy problem. Graham (1972) presents a simple method for sets in R^2. For any finite set $A \subset R^2$, Graham's scan first picks a point in $co(V)$ to be the origin (the average could be used) and sorts all the points lexicographically by polar coordinates relative to that origin. It then scans the point in counterclockwise order, eliminating points not in $co(A)$ and leaving the vertices of $co(V)$ in counterclockwise order. The scan begins with the point $v_1$which is the rightmost point of A and is certainly a vertex of $co(V)$. The scan examines successive triples of consecutive points. Consider, for example, v_1, v_2 and v_3. If v_2 is a vertex of $co(V)$ then $v_1v_2v_3$ must define a reflex angle, that is, the internal angle must be less than 180 degrees. A reflex angle forms a "left turn;" otherwise it is a "right turn." The determinant, Δ, of the matrix

$$\Delta = \begin{vmatrix} v_{21} & v_{22} & 1 \\ v_{11} & v_{12} & 1 \\ v_{31} & v_{32} & 1 \end{vmatrix}$$

tells us if $v_1v_2v_3$ is a left turn. If $\Delta < 0$ the internal angle is a right turn and, because we are scanning in the counter-clockwise direction, the point v_2 is not a vertex and is

eliminated. If v_2 is eliminated Graham's scan next checks $v_1v_3v_4$. Otherwise, Graham's scan then checks $v_2v_3v_4$ to see if v_3 should be eliminated. If v_3 is eliminated, Graham's scan then checks $v_1v_2v_4$ to see if v_2 should be eliminated because if $v_2v_3v_4$ is a right turn then $v_1v_2v_4$ may also be a right turn, putting v_2 inside the face v_1v_4. This checking continues until all successive triples comprise left turns. A more precise description of the algorithm is presented in Graham (1972) and Preparato and Shamos (1989).

There are other methods for approximating convex sets but they are not suitable for our purposes. For example, one may consider Fourier approximation methods to approximate the boundary of $W \subset R^2$ such as in Figure 1. Using a point $w \in W$, such as the point v^N in Figure 1, as the origin, we could use polar coordinates to express the boundary of W. More precisely, let $r(\theta)$ be the distance from v^N to each point P on the boundary of W where θ is the angle Av^NP. Since the boundary of W is continuous, $r(\theta)$ is a continuous periodic function of θ, and has a Fourier series representation, $\sum_{n=1}^{\infty}(a_n \sin n\theta + b_n \cos n\theta)$. We could obtain a finite-dimensional approximation $\widehat{r}(\theta)$ by finding a finite number of points on ∂W, use the data to estimate a finite number of Fourier coefficients a_n and b_n of $r(\theta)$, and use the finite Fourier sum $\widehat{r}(\theta) = \sum_{n=1}^{N}(a_n \sin n\theta + b_n \cos n\theta)$ to approximate ∂W. Unfortunately, the Fourier approximation scheme would not be as useful as a piecewise linear approximation. First, the set defined by $r \leqq \widehat{r}(\theta)$ may not be convex. Second, it would almost surely be neither an inner nor an outer approximation of W since Fourier approximation methods are designed to give approximations which are good on average with the error $r(\theta) - \widehat{r}(\theta)$ alternating between positive and negative values. The same problems would arise for polynomial or spline approximations of $r(\theta)$. Fourier approximations and most standard approximation schemes are unsuitable for our algorithms since the inner and outer nature of our approximations is crucial to various monotonicity properties we exploit.

The methods we presented above are the most suitable ones for our purposes and make the most efficient use of available information. If all you know is that Z is a subset of W, then all you can conclude is that $co\,(Z) \subset W$; no other inference is possible. Furthermore, if you know that Z is a subset of ∂W with corresponding subgradients in G, then all you can conclude is that the outer approximation based on (Z, G) contains W.

4. APPROXIMATIONS OF $B(W)$ AND V

We now combine our convex approximation ideas for sets to construct inner and outer approximations of the B operator. Theory tells us that for any set $W^0 \supseteq V$, the sequence of sets $\{W^k\}_{k=0}^{\infty}$ defined by $W^{k+1} = B(W^k)$ converges to V. Direct application of this technique, however, poses several computational issues. First, we must approximate the convex sets to which we apply $B(.)$. We focus on sets which can be approximated by the methods discussed in the previous section. Second, we need to approximate B. Even if W has a finite representation, its image $B(W)$ possibly involves all pairings of actions with all possible continuation values in W. Section 3 described efficient ways to represent convex sets of infinite points. We now make use of that general discussion to choosing how, given W, we construct $B(W)$ efficiently.

Unfortunately, there is no reason to believe that $B(W)$ is continuous in any topology relevant for computational purposes. This implies that any procedure which tries to approximate B or its fixed points needs to proceed carefully to cope with possible discontinuities. We first define useful kinds of approximation of $B(W)$ which satisfy desirable properties, and then provide details for constructing a few examples of good approximations to $B(W)$.

4.1. Monotone Approximations of $B(W)$. The definition of $B(W)$ in (2) is not an operational definition since it examines an infinite collection of $w \in W$. The critical property of B for our purposes is that it maps convex sets to convex sets and that it is monotone. In particular, $B(W)$ maps the collection of sets $\mathfrak{W} = \{W \subset R^n | W \text{ convex}, W \subset \mathcal{W}\}$ into itself. We are interested in two kinds of approximations of $B(\cdot)$ which preserve critical properties. We define inner and outer monotone approximations of the $B(\cdot)$ operator.

Definition 5. *A mapping $B^I : \mathfrak{W} \to \mathfrak{W}$ is an inner monotone approximation of B if*

1. *for all $W \in \mathfrak{W}$, $B^I(W) \subseteq B(W)$, and*
2. *for all $W, W' \in \mathfrak{W}$, if $W \subseteq W'$ then $B^I(W) \subseteq B^I(W')$*

Definition 6. *A mapping $B^O : \mathfrak{W} \to \mathfrak{W}$ is an outer monotone approximation of B if*

1. *for all $W \in \mathfrak{W}$, $B^O(W) \supseteq B(W)$, and*
2. *for all $W, W' \in \mathfrak{W}$, if $W \subseteq W'$ then $B^O(W) \subseteq B^O(W')$*

The definitions of inner and outer monotone approximations directly imply Lemma 7, which states that B^O and B^I inherit some properties of B.

Lemma 7. *Suppose $B^I(\cdot)$ is an inner monotone approximation of $B(\cdot)$ and $B^O(\cdot)$ is an outer monotone approximation of $B(\cdot)$. Let V be the equilibrium value set. Then the maximal fixed point of B^O contains V and V contains the maximal fixed point of B^I. More precisely,*

1. *if $W_0 \supseteq V$ then $B^O(W_0) \supseteq B^O(B^O(W_0)) \supseteq \cdots \supseteq V$, and*
2. *if $W_0 \supseteq V$ then $B^I(W_0) \supseteq B^I(B^I(W_0)) \supseteq \cdots$, and $V \supseteq \cap_{n=1}^{\infty}(B^I)^n(W_0)$.*

Lemma 7 implies Lemma 8, which presents a sufficient condition for $W \subset V$. In particular, if there is some inner monotone operator B^I such that $W \subset B^I(W)$ then $W \subset V$. This also gives us a sufficient condition for a point w to be an equilibrium value since w is an equilibrium value if $w \in W$ for some W satisfying Lemma 8.

Lemma 8. *Suppose B^I is an inner monotone approximation of B. If $W \subset B^I(W)$ then $B^I(W) \subset B^I(B^I(W)) \subseteq \cdots \subseteq V$.*

The concepts of inner and outer monotone approximations define the critical properties we believe any finite-dimensional approximation to $B(W)$ should satisfy. Lemmas 7 and 8 show that these properties are sufficient for computing reliable approximations to V. We suspect that these properties are also close to being necessary since monotonicity is the key property of $B(W)$ used in theoretical analyses. We now present three examples of approximations to $B(W)$ which satisfy these monotonicity properties.

4.2. Hyperplane Outer Approximation Method. We first use the tangent hyperplane approach to convex set approximations to construct outer monotone approximations of $B(W)$. Since we know that the value set V is convex and that $B(W)$ is convex for any W, we can restrict our analysis to convex W. The key to approximating $B(W)$ is to fix some subgradients $H \subset R^N$ (we call them *search subgradients*) and locate boundary points x of $B(W)$ where the subgradient of $B(W)$ at x is in H. The idea is to define an outer monotone approximation of B, $B^O : \mathfrak{W} \to \mathfrak{W}$, which takes convex polytopes as input and produces a convex polytope which surely contains $B(W)$. We then iterate B^O until successive iterates are close. By monotonicity, this iterate contains V. Since the approximation of $B(W)$ is distinct from the computation of V, we discuss them separately.

Algorithm 1: Outer Monotone Approximation of $B(W)$

Parameters: Search subgradients: L subgradients $H = \{h_1, ..., h_L)\} \subset R^N$

Input: Description of W: Approximation subgradients, $G = \{g_1, ..., g_M)\} \subset R^N$, and levels, $C = \{c_m | m = 1, .., M\} \subset R$ such that $W \equiv \cap_{\ell=1}^{M} \{z \mid g_m \cdot z \leqq c_m\}$.

Step 1: Find extremal points of $B(W)$. For each $h_\ell \in H$:

(a) For each $a \in A$, find optimal feasible equilibrium value in the h_ℓ direction assuming that action a is the current action profile; that is

$$\begin{array}{rcl} c_\ell(a) & = & \max_w \; h_\ell \cdot [(1-\delta)\Pi(a) + \delta w] \\ & & (i) \; w \in W \\ & & (ii) \; (1-\delta)\Pi^i(a) + \delta w_i \geq \\ & & \qquad (1-\delta)\Pi_i^*(a_{-i}) + \delta \underline{w}_i, \; i = 1, .., N \end{array} \tag{6}$$

where $c_\ell(a) = -\infty$ if there is no w satisfying the constraints in (6).

(b) Choose best action profile $a \in A$, by computing $c_\ell^+ = \max \{c_\ell(a) | a \in A\}$

Output: $B^O(W; H) = W^+$, where levels are collected in $C^+ = \{c_1^+, \cdots, c_L^+\}$, the set of approximation subgradients for W^+ is H, and $W^+ = \cap_{\ell=1}^{L} \{z \mid g_\ell \cdot z \leqq c_\ell^{k+1}\}$

Algorithm 1 presents the outer approximation hyperplane method for approximating $B(W)$. We first specify the search gradients H which are parameters of the approximation. The approximation presumably improves as we use larger sets H but the algorithm will take longer. The input of Algorithm 1 must be a finitistic description of W. For this algorithm, this description is a list of gradients (which we will call *approximation subgradients* since they approximate the inputted set W) and levels..

The key computation occurs in Step 1. Specifically, for each search subgradient h_ℓ and action profile a, Step 1a finds a continuation value $w \in W$ which makes a incentive compatible and maximizes a weighted sum of player payoffs where the weights are given

by h_ℓ; if there is no such w then we set the value of the problem to be $-\infty$. Examination of (6) shows that it is a linear programming problem. The objective is linear in w, the choice variable. Since a is fixed, the incentive compatibility constraints in (ii) are linear in the continuation payoff w. Since the input W is a convex polytope, (i) is also a set of linear constraints using the inputted approximation subgradients G and hyperplane levels C to describe the constraint $w \in W$. Therefore, (6) is a linear programming problem. This allows us to exploit existing linear programming code, which essentially checks a finite number of vertices to find the maximum. Therefore, the approximation error in solving (6) is the same order as machine epsilon, which is about 10^{-16} on most current machines using double precision arithmetic.

Step 1a is executed for each $h_\ell \in H$ and action profile $a \in A$. For each $h_\ell \in H$, Step 1b picks that action profile, a^*_ℓ, which produces the largest weighted payoff, denoted c^+_ℓ, in Step 1a. Suppose that w^*_ℓ is the continuation value supporting a^*_ℓ. The construction in Step 1 implies that the hyperplane $h_\ell \cdot z = c^+_\ell$ is tangent to $B(W)$ at $w = (1-\delta)\Pi(a^*_\ell) + \delta w^*_\ell$, and that every point in $z \in B(W)$ satisfies $g_\ell \cdot z \leqq c^+_\ell$.

Step 1 constructs such a hyperplane for each search subgradient $h_\ell \in H$. The output step then collects the hyperplanes constructed in Step 1 to construct a description of the output, W^+. Since the set of search subgradients in Step 1 is H, the approximation subgradient set for W^+ is H. The hyperplane levels are collected in C^+. The set W^+ defined in Step 2 is the output $B^O(W;H)$ and contains $B(W)$. $B^O(W;H)$ is clearly monotone in W. Therefore, $B^O(W,H)$ is an outer monotone approximation of $B(W)$. It is also antimonotone in H in that if $H \subset H'$ then $B^O(W;H') \subset B^O(W;H)$. Lemma 9 summarizes the key properties of $B^O(W;H)$.

Lemma 9. *For any set of subgradients $H \subset R^N$,*

1. *$B^O(W;H)$ is an outer monotone approximation of $B(W)$;*
2. *if $H \subset H'$ then $B^O(W;H') \subset B^O(W;H)$*

Proof. The equations defining Algorithm 1 show that it computes points on the boundary of $B(W)$ with subgradients H. Since Step 2 uses an outer approximation construction, $B^O(W;H) \supset B(W)$. If $W' \subset W$, the only part of (6) which differs for $W' \neq W$ is the list of constraints implicit in (i). Since they are tighter for $W' \subset W$, the solution for W' will produce smaller $c_\ell(a)$ results, smaller values of c^+_ℓ which in turn,

since the approximation subgradients of W^+ are unaffected, produce smaller output sets. Therefore, $B^O(W;H) \subset B^O(W';H)$. Part 2 is similarly obvious. ■

We next use our $B(W;H)$ approximations in an algorithm to produce an outer approximation of V. Algorithm 2 displays the outer hyperplane procedure. Step 0 initializes the problem by choosing a set of search subgradients H and an initial guess W^0.

Algorithm 2: Outer Hyperplane Algorithm for Approximating V

Step 0: Initialize elements and construct initial guess $W^0 \supset \mathcal{W}$

(a) Subgradients: Choose L search subgradients $H = \{h_1, ..., h_L)\} \subset R^N$

(b) Points: Select boundary points $Z^0 = \{z_1^0, \cdots, z_L^0)\} \subset R^N$.

(c) Levels: compute hyperplane levels $c_\ell^0 = g_\ell^0 \cdot z_\ell^0,\ \ell = 1, .., L$, and collect levels in C^0

(d) Define W^0: $W^0 = \cap_{\ell=1}^L \{z \mid g_\ell \cdot z_\ell \leqq c_\ell^0\}$

Step 1: Construct $W^{k+1} = B^O(W^k;H)$ where $C^{k+1} = \{c_1^{k+1}, \cdots, c_L^{k+1}\}$ define W^{k+1} in $W^{k+1} = \cap_{\ell=1}^L \{z \mid g_\ell \cdot z \leqq c_\ell^{k+1}\}$

Step 2: Stop if W^{k+1} is close to W^k. Specifically, stop if $\max_\ell \left|c_\ell^{k+1} - c_\ell^k\right| < \epsilon$; else go to 1.

Step 0 constructs an initial guess which contains $\mathcal{W}$ which contains V. Step 1 is the key iteration, computing successive iterates of $B(W;H)$. Algorithm 2 checks the stopping criterion, which is a key part of any iterative algorithm. After the initial guess, each face of each iterate has a normal in H, and each iterate will be of the form $\cap_{\ell=1}^L \{z \mid g_\ell \cdot z_\ell \leqq c_\ell\}$. The stopping criteria in Step 2 measures the difference between W^k and W^{k+1} by computing the differences in the levels c_ℓ^k. If the levels in C^k and C^{k+1} are close then W^k and W^{k+1} are close in the Hausdorff metric since the faces of W^k and W^{k+1} are parallel and the differences between C^k and C^{k+1} equal the distances between the parallel faces. We stop the algorithm when the maximal change in the c_ℓ is small. In our examples

below we set $\epsilon \geq 10^{-5}$, choices which produce four-digit or greater accuracy. This may be more demanding than some require, but our goal with these examples is to show the reader the kinds of problems which this algorithm can accurately handle. Many possible improvements on Algorithm 2 are suggested by nonlinear equation theory. For example, we could use Gauss-Seidel methods and still maintain all features of the outer hyperplane method. We leave the development of these ideas to future work.

Lemma 10 shows that the outer hyperplane approximation method does stop at some finite time. This is an important property not true of many numerical methods, such as Newton's method.

Lemma 10. *The outer hyperplane method will stop at some finite iteration and produce a set which contains V.*

Proof. Since H is the approximation subgradient set for all iterates, Algorithm 2 defines a function $\Psi : R^L \to R^L$ mapping the hyperplane levels C^k for W^k to hyperplane levels C^{k+1} for W^{k+1}. The iterates of Algorithm 2 form a sequence of closed sets, each iterate a closed subset of the previous iterate. Therefore, the sequence of sets converges to a limit set in the Hausdorff norm. Similarly, the iterates C^k are monotone in that $c_\ell^k < c_\ell^{k+1} (c_\ell^k > c_\ell^{k+1}) \Rightarrow c_\ell^{k+1} < c_\ell^{k+2} (c_\ell^{k+1} > c_\ell^{k+2})$ which in turn implies that each c_ℓ^k sequence converges monotonically to some limit. Therefore, for any $\epsilon > 0$, the maximal change in the c_ℓ levels will be less than ϵ. The final iterate contains V since the initial guess contains V and $B^O(W; H)$ is an outer monotone approximation of $B(W)$. ■

We need to be clear about the proper uses of the outer hyperplane approximations of V produced by the outer hyperplane algorithm. The final iterate, W^*, will contain V. It may not be a good approximation of V since W^* may contain points not in V. Therefore, we can only conclude that if w is not in W^* then w is not an equilibrium value of the supergame. We cannot say anything about any point in W^*. The inner approximation methods presented below are necessary to produce points which are equilibrium values.

A similar outer hyperplane approximation method is examined in Cronshaw (1997), who generalizes Conklin and Judd (1993) by including continuous strategy spaces. The generalization to continuous strategies is clear since the key optimization problem for each

h_ℓ in Step 1 is really

$$\begin{aligned} c_\ell \quad = \quad & \max_{w,a} \; h_\ell \cdot [(1-\delta)\Pi(a) + \delta w] \\ & (i) \; w \in W^k \\ & (ii) \; (1-\delta)\Pi^i(a) + \delta w_i \geq (1-\delta)\Pi_i^*(a_{-i}) + \delta \underline{w}_i, \; i = 1, .., N \end{aligned} \tag{7}$$

The optimization problem in (7) is nonlinear because a is free, but it is a linear programming problem for a fixed a. Our Step 1 exhaustively examines all possible action profiles $a \in A$ and then picks the best result. This exhaustive procedure will find the true solution but is not possible with continuous actions. Therefore, the Cronshaw scheme may solve (7) with some nontrivial error. Since the objective in (7) can be a complicated nonlinear function of a, no optimization scheme can be relied on to find the true solution. When there is a continuum of actions a, solutions to (7) will have some numerical error. Therefore, the approximation of $B(W)$ may miss some points inside the true $B(W)$, and violate the outer monotone approximation property. This can be a particularly vexing problem in games since there may be multiple local solutions to (7), forcing one to use global optimization methods. There may be cases where one can reliably compute solutions to (7), but that would necessarily rely on special properties of the game. Since we want to present an algorithm applicable to any game and be sure that iterations of the approximation to $B(W)$ will always contain V, we stay with finite actions. Cronshaw also uses his algorithm to compute equilibrium strategies. This is somewhat speculative since we only know that the output contains V, not that the boundary points of the outer approximation are in V. Therefore, Cronshaw's computations of equilibrium actions and strategies may not be valid. Since we want to compute some equilibrium values and actions, we next present inner approximation methods.

4.3. Inner Hyperplane Approximation Method. We next use convex set approximation methods to construct an inner approximation to $B(W)$ for convex polytopes W and an inner approximation to V. The main part of the algorithm is the same as the outer approximation; the difference lies in the way the convex set is constructed in each iteration and the information we need to record at each step. For the outer approximation, the faces of the outer approximation to $B(W)$ are the tangent hyperplanes. For the inner approximation, the approximation to $B(W)$ is the convex hull of the boundary points found in Step 1 of Algorithm 1, and the faces must be constructed by a procedure

which constructs a convex hull. We then use inner approximations of $B(W)$ to construct an inner approximation of V.

Algorithm 3 defines the monotone inner approximation $B^I(W;H)$. We first choose the set H of search subgradients. The inputs for Algorithm are the inputs for Algorithm 1 plus a set of vertices Z such that $W = co(Z)$. Step 1a of Algorithm 3 also solve (6) and records both the maximized objective $c_\ell(a)$ and the maximizing choice of continuation value, $w_\ell(a)$. Step 1b then finds that action which maximizes the h_ℓ-weighted payoffs, and records both the maximizing action, a^*_ℓ, as well as the maximal weighted payoff, z'_ℓ. Step 1c collects the z'_ℓ points into Z', which are all in $B(W)$. Since we want an inner approximation, we can only conclude that $co(Z') \subset B(W)$. Therefore, $co(Z')$ is our inner approximation, which we call W^+.

Step 1 describes W^+ in terms of Z'. This is not an adequate representation since we often need to describe W^+ in terms of its faces. Therefore, Step 2 first eliminates those vertices of Z' which are not vertices of W^+. In R^2 we use Graham's scan to do this. The result, Z^+, is the set of vertices of W^+. Step 2 then computes the approximation subgradients, G^+, and hyperplane levels, C^+, which define the faces of W^+. In R^2 this is trivial once Graham's scan has produced the vertices in counterclockwise order, but is a more substantive problem in general. The output consists of the sets G^+, C^+, and Z^+. Algorithm 3 essentially takes input (G, C, Z) and produces output (G^+, C^+, Z^+). The input Z is not necessary for computing $B(W;H)$ but we keep the vertices since the convergence criterion in Algorithm 4 needs them.

Algorithm 3: Monotone Inner Hyperplane Approximation of $B(W)$

Parameters: Choose L search subgradients, $H = \{h_1, ..., h_L)\}$

Input: Description of W: Approximation subgradients, $G = \{g_1, ..., g_M)\} \subset R^N$, and levels, $C = \{c_m | m = 1, .., M\} \subset R$ such that $W \equiv \cap_{\ell=1}^{M} \{z \mid g_m \cdot z \leqq c_m\}$, and vertices $Z = \{z_1, \cdots, z_M)\}$ such that $W = co(Z)$.

Step 1: Find extremal points of $B(W)$. For each search subgradient $h_\ell \in H$:

(a) For each $a \in A$, solve

$$\begin{aligned} c_\ell(a) \quad = \quad & \max_w \; h_\ell \cdot [(1-\delta)\Pi(a) + \delta w] \\ & (i) \; w \in W \\ & (ii) \; (1-\delta)\Pi^i(a) + \delta w_i \geq \\ & \qquad (1-\delta)\Pi_i^*(a_{-i}) + \delta \underline{w}_i, \; i = 1, .., N \end{aligned} \tag{8}$$

where $c_\ell(a) = -\infty$ if there is no w satisfying the constraints. Let $w_\ell(a)$ be a w value which solves (8).

(b) Compute value of best action profile $a \in A$ and corresponding continuation value:

$$\begin{aligned} a_\ell^* \quad &= \quad \arg\max \{c_\ell(a) | a \in A\} \\ z'_\ell \quad &= \quad (1-\delta)\Pi(a_\ell^*) + \delta w_\ell(a_\ell^*) \end{aligned}$$

(c) Collect set of vertices of convex hull: $Z' = \{z'_\ell | \ell = 1, ..., L\}$, and define $W^+ = co(Z')$.

Step 2: Compute $Z^+ = \{z' \in Z' | z' \in \partial W^+\}$, and find subgradients $G^+ = \{g_1^+, ..., g_{M^+}^+\}$ and constants $C^+ = \{c_1^+, ..., c_{M^+}^+\}$ such that $co(Z^+) = \cap_{\ell=1}^{M^+} \{z \mid g_m^+ \cdot z \leqq c_m^+\} = W^+$

Output: The result $B^I(W; H) = W^+$ is represented by approximation subgradients, G^+, levels C^+, and vertices Z^+.

Lemma 11 presents the basic properties of Algorithm 3. The operators $B^I(W;H)$ and $B^O(W;H)$ are similar except that $B^I(W;H)$ is monotone increasing in H.

Lemma 11. *For any set of subgradients $H \subset R^N$,*

1. *$B^I(W;H)$ is an inner monotone approximation of $B(W)$;*
2. *if $H \subset H'$ then $B^I(W;H) \subset B^I(W;H')$*

Proof. Part 1 follows from the same arguments made in Lemma 9. Part 2 follows from the observation that increasing the number of search gradients in Algorithm 3 increases the number of vertices computed, which enlarges the convex hull of the computed vertices. ■

Our inner approximation algorithm for approximating V is presented in Algorithm 4. It first fixes the search subgradients H and then iterates $B^I(W;H)$. As in Algorithm 2, we need to define a concept of two sets being close for a stopping rule, but we cannot just look at the C^k sets since the approximation subgradients for W^k and W^{k+1} can be different. We use a norm based on the vertices. If $W_i = co(Z_i)$, $i = 1, 2$, where the Z_i are finite sets, then the distance between W_1 and W_2 is defined to be

$$d(Z_1, Z_2) = \max\left\{\max_{z_1 \in Z_1}\min_{z_2 \in Z_2}\|z_1 - z_2\|,\ \max_{z_2 \in Z_2}\min_{z_1 \in Z_1}\|z_1 - z_2\|\right\}$$

The Hausdorff distance between $co(Z_1)$ and $co(Z_2)$ is bounded above by $d(Z_1, Z_2)$ because each face of $co(Z_i)$ is a convex combination of the vertices in Z_i. Therefore, two sets close in $d(.,.)$ is be close in the Hausdorff metric. The details of the algorithm for the inner approximation are in Algorithm 4.

Algorithm 4: Inner Hyperplane Approximation Algorithm for V

Step 0: Initialize elements:

(a) Search subgradients: Choose L search subgradients $H = \{h_1, ..., h_L)\}$.

(b) Vertices: Select vertices, $Z^0 = \{z_1^0, \cdots, z_M^0\}$, for initial guess $W^0 \equiv co(Z^0)$.

(c) Hyperplanes: Define the gradients, $G = \{g_1^0, ..., g_M^0)\}$, and levels, $C^0 = \{c_m^0 | m = 1, .., M\}$, which define W^0 in $W^0 = \cap_{\ell=1}^{M} \{z \mid g_m^0 \cdot z_m^0 \leqq c_m^0\}$.

Step 1: Construct $W^{k+1} = B^I(W^k; H)$ with vertices $Z^{k+1} = \{z_1^{k+1}, \cdots, z_M^{k+1}\}$.

Step 2: If $d(Z^{k+1}, Z^k) < \epsilon$, set $W^{I*} = W^{k+1}$ and stop; else set $k = k + 1$ and go to Step 1.

Lemma 12 states the key properties of the inner hyperplane method.

Theorem 12. *The inner hyperplane method for V will stop at some finite iteration. The limit point of the iterations $W^{k+1} = B^I(W^k; H)$ is a subset of V.*

Proof. The first claim follows from the same arguments used in Theorem 10. The second claim is true since the limit of $\left(B^I\right)^k (W; H)$ is a fixed point of $B^I(W; H)$ and $B^I(V; H) \subset B(W)$. ■

The result for the inner hyperplane method is not as good as it was for the outer hyperplane method since the output of Algorithm 4 may contain points not in V. This is not satisfactory since we want to construct a set which we can prove to be a subset of V. Corollary 13 presents a computable sufficient condition for $W^k \subset V$ for some k in Step 1 of Algorithm 4. It follows from Lemmas 8 and 11 and will be used below to construct subsets of V.

Corollary 13. *Consider the sequence $W^{k+1} = B^I(W^k; H)$. If for some k $W^k \subset W^{k+1}$, then for all $k' > k$, $W^{k'} \subset V$.*

4.4. Inner Ray Approximation Method. The hyperplane procedures produce inner and outer approximations of the equilibrium value set, but they produce limited information about the actions and strategies which support those values. We often want to know equilibrium actions and strategies that support particular points in V to understand an equilibrium, information not provided by the hyperplane procedures. Also, we used a public randomizing device to make the equilibrium value sets convex. We often want to know if the lotteries are part of an equilibrium or if an equilibrium value can be supported without randomization.

We introduce a *ray procedure* which takes a set W and produces an inner approximation, W', of $B(W)$ together with information about the actions $a \in A$ and continuation values in W which will support particular values of interest in W', not just at the vertices if W'. Our ray procedure begins with an a priori known value $w^0 \in B(W)$; a natural generic choice would be a Nash equilibrium value v^N, but the geometry of $B(W)$ may suggest better choices for particular games. This is the weakness of the ray method since it may be difficult to find a good choice for w^0 which allows $B(W)$ to be approximated efficiently. The hyperplane approach is better in this regard. However, the ray method will be able to find actions and strategies corresponding to particular points on $\partial B(W)$, a task which hyperplane methods do not accomplish. Therefore, we need to develop both methods.

The ray procedure uses the fact that any ray originating from $w^0 \in B(W)$ intersects the boundary of $B(W)$ exactly once. Algorithm 5 presents the inner ray procedure for approximating $B(W)$. We want to approximate $B(W)$ in the same way that W is approximated in Figure 1, where w^0 here plays the role of v^N in Figure 1. Therefore, the first step chooses the parameters w^0 and the points M on a sphere which define the rays, just as the points A', B', etc., defined rays from the origin v^N in Figure 1. The inputs for Algorithm 5 are the same as for the inner hyperplane approximation method in Algorithm 3.

Algorithm 5: Inner Ray Approximation of $B(W)$

Parameters: Origin, $w^0 \in B(W)$, and M points, $\Theta \subset R^N$, on the unit sphere.

Input: Description of W: Approximation subgradients, $G = \{g_1, ..., g_M)\} \subset R^N$, and levels, $C = \{c_m | m = 1, .., M\} \subset R$ such that $W \equiv \cap_{\ell=1}^{M} \{z \mid g_m \cdot z \leqq c_m\}$, and vertices $Z = \{z_1, \cdots, z_M)\}$ such that $W = co(Z)$.

Step 1: Find extremal points of $B(W)$. For each $\theta_\ell \in \Theta$:

(a) For each action profile $a \in A$ solve

$$\begin{array}{rcl} \lambda_\ell(a) & = & \max_{\lambda, w} \ \lambda \\ & & \text{(i) } w = \delta^{-1}[(w^0 + \lambda\theta_\ell) - (1-\delta)\Pi(a))] \in W \\ & & \text{(ii) } (1-\delta)\Pi^i(a) + \delta w_i \geq (1-\delta)\Pi_i^*(a_{-i}) + \delta \underline{w}_i, \ \forall i \\ & & \text{(iii) } \lambda \geq 0 \end{array} \tag{9}$$

where $\lambda_\ell(a) = -\infty$ if there is no w satisfying the constraints.

(b) Set $\lambda_\ell^* = \max_{a \in A} \lambda(a)$, $a_\ell^* = \arg\max_{a \in A} \lambda(a)$, and $z'_\ell = w^0 + \lambda_\ell^* \theta_\ell$.

(c) Collect set of vertices of convex hull: $Z' = \{z'_\ell | \ell = 1, ..., L\}$, and define $W^+ = co(Z')$

Step 2: Compute $Z^+ = \{z' \in Z' | z' \in \partial W^+\}$, and find subgradients $G^+ = \{g_1^+, ..., g_{M^+}^+\}$ and constants $C^+ = \{c_1^+, ..., c_{M^+}^+\}$ such that $co(Z^+) = \cap_{\ell=1}^{M^+} \{z \mid g_m^+ \cdot z \leqq c_m^+\} = W^+$

Output: The result $B^I(W; H) = W^+$ is represented by approximation subgradients, G^+, levels C^+, and vertices Z^+. The point $z'_\ell \in Z^+$ on the ray $\overrightarrow{w^0\theta_\ell}$ is supported by action profile a_ℓ^* and continuation value $\delta^{-1}[z'_\ell - (1-\delta)\Pi(a_\ell^*)]$.

Step 1 in Algorithm 5 examines, for each $\theta \in \Theta$, the direction $\overrightarrow{w^0\theta}$. It finds the intersection of $\overrightarrow{w^0\theta}$ with $\partial B(W)$ by maximizing λ along the line $w^0 + \lambda\theta_\ell$ subject to $\lambda \geq 0$ and the constraints in (9). If $w' \in B(W)$ equals $w^0 + \lambda\theta_\ell$ and is supported by a current action a and continuation value w, then $w = \delta^{-1}[(w^0 + \lambda\theta_\ell) - (1-\delta)\Pi(a))]$ is

a linear restriction on w and λ. Since the input describes W as an intersection of half-planes, the condition $w \in W$ is described by set of constraints linear in w. The incentive compatibility constraints in (ii) are linear in w. Therefore, (9) is a linear programming problem. For some a there will be no solution since no promise $w \in W$ can support action a today; in those cases we set $\lambda_\ell(a) = -\infty$. Since $w^0 \in B(W)$ there will always be some a such that (9) has a meaningful solution (even if it is $\lambda = 0$). Step 1b finds the $a \in A$ yielding the maximal $\lambda_\ell(a)$, denoted λ_ℓ^*, along with a supporting action a_ℓ^* and a point z'_ℓ in $B(W)$. We do this for each direction $\theta_\ell \in \Theta$. Step 2 then defines the next iterate to be the convex hull of the z'_ℓ points constructed in Step 1, and records the actions and supporting continuation values. After the vertices are determined, the inner ray approximation proceeds by finding the piecewise linear functions that define the boundary of the new set.

Iterations of the ray procedure in Algorithm 5 can be used to produce an inner approximation to V. The procedure follows Algorithm 4 except to use the ray method in Algorithm 5 for $B^I(W; H)$ instead of the inner hyperplane method of Algorithm 3; we will call this the *inner ray approximation algorithm for* V. Since the details are clear, we do not present a formal listing here. This was the method initially developed in Conklin and Judd (1993).

4.5. Computing Actions and Strategies. The ray method for computing an inner approximation has the additional advantage of also providing the actions and continuation values for any point of $B(W)$. We now indicate precisely how we derive actions and strategies once we have found an inner approximation. Suppose that we have found $W^I \subset V$ and want to determine how a certain point $x \in \partial W^I$ is supported. We then choose some point w^0 in the interior of W^I and apply Step 1 of Algorithm 5 using the ray defined by $w^0 + \lambda(x - w^0)$ for $\lambda > 0$, and record the action profile and continuation value that yields the weighted maximum payoffs to the players. If the maximal point is x, then x is supported by a pure strategy; otherwise, $x \in \partial W^I$ because it is a convex combination of vertices of W^I which are supported by pure strategies. Of course, there may be multiple solutions; our programs make no attempt to list all strategies. Since one can also identify the actions associated with worst punishment payoffs, strategies can be constructed. In the subsequent sections with specific application of our algorithms to familiar games, we also provide sample action paths and strategies constructed in this

manner.

The inner hyperplane method can also determine action choices at the vertices of $B(W;H)$. As in the ray method one simply keeps record of the action pair that yields the highest discounted payoff, weighted by the normal vector. However, the equilibrium values determined by the hyperplane method are solutions to linear programming problems and these solutions always correspond to the vertices of the polygon being estimated. Therefore even for different search subgradients, or weights on players discounted payoff, the equilibrium values computed are often the same, rendering it difficult to examine actions associated with points on $\partial B(W;H)$ other than the vertices. For example, with the ray method we are able to identify the actions that correspond to the symmetric equilibrium value on the efficient frontier of a value set (that is, the point on $\partial B(W;H)$ on the 45 degree line), whereas the hyperplane method can do this only if that point is a vertex of $B(W;H)$. It is therefore advantageous to use the ray method for determining sample action paths and strategies.

The ray procedure has two problems. First, it requires that we know a point in V. This is not a problem for the static games we consider here, but it is a problem for extensions such as games with state variables where computing any dynamic Nash equilibrium may be as difficult as solving for the equilibrium value set of all Nash equilibria. Second, the one point which we know to be in V may be a poor choice around which to build an approximation of V. For example, if v^N is the minmax value, then v^N will lie on the boundary of V and many rays from v^N to ∂V will have length zero, and Step 1 in Algorithm 5 will be pointless for many rays $\theta \in \Theta$. For these reasons, we emphasize the role of inner and outer hyperplane approximations of the set V and focus on the role of the inner ray procedures for computing equilibrium actions and strategies.

4.6. Algorithm for Approximating V and Equilibrium Strategies. We have discussed three algorithms which accomplish three distinct tasks. The convex polytope algorithm in Algorithm 6 integrates these three simple algorithms into a complete convex polytope algorithm for approximating V. We use $\mathcal{W}$, the largest possible set of equilibrium values, as our initial guess. We next choose the search subgradients, H, used by the outer and inner hyperplane approximations of $B(W)$. We use a uniformly distributed set of gradients since we do not know have any information indicating any particularly useful subgradients. We also choose a stopping criterion; one should try a loose criterion first and

then tighten it if the final error bound exceeds some acceptable level. Step 2 computes the outer approximation, W^O, by iterating $B^O(W;H)$ until the stopping criterion is satisfied. We then use W^O as the initial guess for iterations of $B^I(W;H)$ until the stopping criterion is satisfied. The result, W^{II}, is not necessarily an inner approximation since the iterates of $B^I(W;H)$ may never have satisfied Corollary 13. We then execute a "bounce step," looking for a set which satisfies the assumptions in Lemma 8. We do this by reducing W^{II} and then see if it bounces back when we compute $B(W^{II})$. This is the one place where there is no surefire way to proceed. We were successful in all of our examples, but this may have been a matter of luck. In particular, we shaved off a small area in the southeast and/or northwest corners of the Pareto frontier of W^{II} to construct a slightly smaller set, W', and found that $B^I(W';H) \supset W^{II}$. Our experience indicated that a key factor in finding a good W' was keeping the threat points of W^{II}. Step 6 then iterates an inner approximation method until the stopping criterion is satisfied. The result, W^I, is an inner approximation of V. Step 7 then chooses some set W' in W and uses the ray method to find strategies supporting those points.

Algorithm 6: Convex Polytope Algorithm for V

Step 1: Initial Guess: Choose $W^0 = \mathcal{W}$.

Step 2: Parameters: Choose search subgradients H and stopping criterion ϵ.

Step 3: Outer Approximation: Iterate $W^{k+1} = B^O(W^k; H)$ until convergence criterion is satisfied. Output is the outer approximation W^O.

Step 4: Intermediate Inner Approximation: Set $W^0 = W^O$. Iterate $W^{k+1} = B^I(W^k; H)$ until convergence criterion is satisfied. Output is W^{II}.

Step 5: Bounce Step: Find some $W' \subset W^{II}$ such that $B^I(W'; H) \supset W^{II}$.

Step 6: Inner Approximation: Set $W^0 = W'$. Iterate $W^{k+1} = B^I(W^k; H)$ until convergence criterion. Output is W^I.

Step 7: Compute Strategies: Pick some points $W' \subset W^I$ and use the ray method to compute strategies supporting W'.

Algorithm 6 produces a final result similar to that displayed in Figure 4. If V is the true solution to a game, then W^O is like the polygon $ABCDEFGH$ and contains V, and W^I is like the polygon $abcdefgh$ inside V. The dark shaded region between W^O and W^I represents a region of uncertainty since we do not know if a point there is in V or not, but is also represents an error bound on our approximations to V since we know that the boundary of V is somewhere in W^O/W^I.

The results of our convex polytope algorithm include outer and inner approximations of V, an error bound, W^O/W^I, and actions which support points in the inner approximation. Other procedures do some of these tasks but not all. For example, Cronshaw (1997) only computes an outer approximation.[3] We know of no work which discusses an error bound. In summary, our convex polytope method offers an integrated approach that performs several useful tasks.

[3]Others have also discussed computational methods for similar problems but the paucity of details offered in their papers make it difficult to ascertain what they do, and impossible for us to compare our algorithms with theirs.

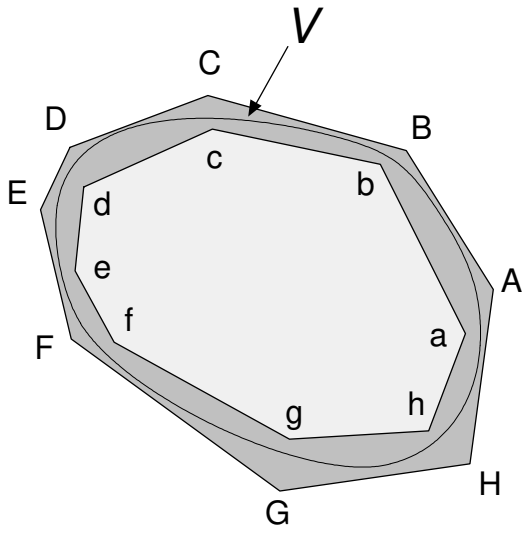


Figure 4: Typical error bounds

5. Applications

We now apply our algorithms to some familiar games to illustrate our algorithms' critical features. Since these games are simple, we can ascertain their value sets without using our algorithms and then compare the results of our algorithms to what we know about the value sets. This allows us to demonstrate the high quality approximations produced by our convex polytope methods and indicate how much effort is necessary. This follows the standard approach in numerical analysis of testing an algorithm by applying it to known solutions of well-understood problems[4]. We examine Prisoner's Dilemma, Battle of the Sexes, Cournot duopoly, and Bertrand duopoly. Our primary aim is to quantitatively demonstrate the properties of the recursive value set algorithm, including monotonicity of the $B(W)$ operator stated in Theorem 1, and the dependence of V on the discount factor δ. We also demonstrate how the precision of our approximation is affected by the fineness of the approximation and the stopping criterion.

[4]Some readers may prefer to see the results of our algorithm applied to more interesting games. We invite them to get our programs from us and apply them to their favorite games. In this paper, it is more appropriate for us to use familiar games to demonstrate the properties of our algorithms.

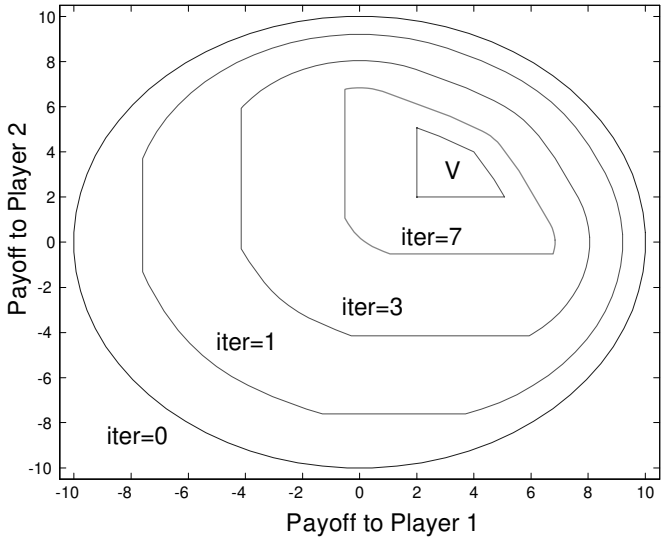


Figure 5: Prisoner's Dilemma: Convergence

5.1. Prisoner's Dilemma. Our first example is a Prisoner's dilemma game. Each player chooses either cooperate (C) or defect (D) and payoffs are

		Player 2: C	Player 2: D
Player 1:	C	4,4	0,6
	D	6,0	2,2

We first use this game to demonstrate the monotonicity of our outer approximations to the $B(W)$ operator. Figure 5 assumes the discount rate $\delta = .8$ and displays some early iterates W^k from the outer hyperplane approximation of $B(W)$ along with the final solution V. The iterates are nested and become smaller, converging to V. The rate of convergence is similar to the discount factor $\delta = .8$; larger δ implies slower convergence here just as it does in the related value function iteration method for dynamic programming.

Polygon $ABCDEF$ in Figure 6 represents both the inner and outer approximations to our prisoner's dilemma game with $\delta = .8$. It was generated using with 72 uniformly distributed subgradients, and the difference between the inner and outer approximations cannot be seen in Figure 6. If the set of gradients is too small, there may be large errors. Figure 6 depicts the outer approximation of V with three different sets of gradients. The

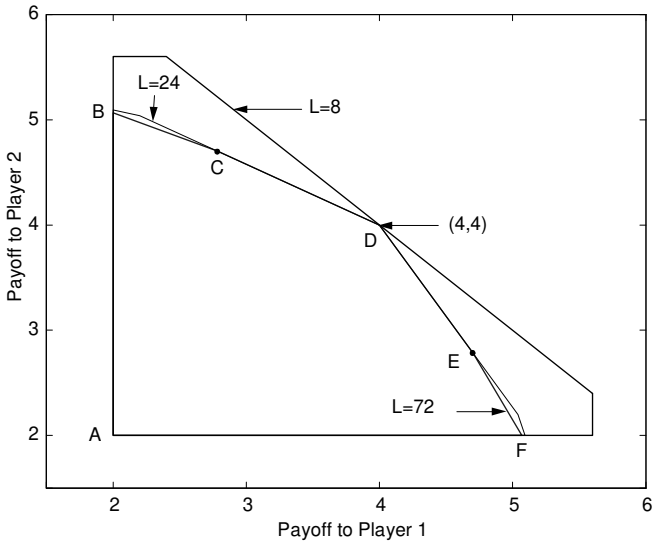


Figure 6: Prisoner's Dilemma using different subgradients.

outermost lines represent the value set after convergence of the outer hyperplane method when we used 8 equispaced gradients, and the set of lines slightly outside of the solution $ABCDEF$ represents the case of 24 gradients. The outer approximation with only 8 gradients includes many values which are not part of the 24- and 72-gradient cases, and, hence, are not part of the equilibrium. Since there is no good rule for determining how many gradients are necessary, we should continue to add gradients until the differences between the inner and outer approximations are small. In this case we knew that the $L = 8$ and $L = 24$ outer approximations were not adequate because they were too far from the inner approximations (not illustrated here). We continued to increase L until the outer and inner approximations were indistinguishable in Figure 6. More precisely, the difference between the final inner and outer approximations were less than 10^{-3}.

Next we examine how the algorithm performs as we change the discount factor, δ. This game has simple properties—e.g., the unique one-shot Nash equilibrium is also the minmax payoff for both players— that allow us to determine some things directly. In our Prisoner's Dilemma game we know that cooperation, (C,C), cannot be sustained for discount factors less than 0.5. Our algorithms easily find this, collapsing to the singleton set consisting of the Nash value after only a few iterations. The minimum discount factor

required to sustain combinations of cooperation at (C,C) and asymmetric plays at (C,D) is less obvious. Direct computations show that for $\delta \geqslant 0.5$ asymmetric outcomes such as (5,2) and (2,5) can be supported. For example, (5,2) can be supported by playing (D,C), repeating (D,C) until the incentive compatibility condition of Player 2 binds, and then switching to (C,C) forever. This is a good test for the inner approximation step since for δ just barely above 0.5 the inner approximations may miss some critical point on the boundary of V and collapse to the Nash equilibrium. Even for a discount factor as low as 0.501, our inner approximation algorithms, using 8 or more subgradients did not collapse and did find the strategies which support the vertices (5,2) and (2,5).

5.2. Battle of the Sexes. We next examine a 2×2 Battle of the Sexes game with payoffs

		Player 2: B	Player 2: P
Player 1:	B	8,5	3,3
	P	0,0	5,8

The players want to coordinate their choices by both choosing B or both choosing P, but disagree over how to coordinate. Figure 7 displays the computed equilibrium value sets for Battle of the Sexes for $\delta = 0.8$ and 0.571.

V obviously includes the line between (5,8) and (8,5), the two pure-strategy one-shot Nash equilibria. If $\delta \in (4/7, 1]$, the minmax value for each player is 5, because in some equilibria, with public randomization, the payoff of (5,5) can be realized using convex combinations of (3,3) along with (5,8) and (8,5). With sufficient patience each player is willing to mete out or accept punishments that yield an average present value of (5,5). We can verify these computations analytically. The minmax point (5,5) is supported by a strategy starting with the action profile (B, P). (B, P) gives a one-shot payoff of (3,3). Given that the maximum either player can get from deviating is 5, we can compute the implied continuation value promise offered to the players by the incentive compatibility constraint, $(1-\delta)3+\delta v_i \geq (1-\delta)5+\delta 5$, implying $\delta(v_i-3) \geq 5-3$. The greater v_i is, the less δ needs to be in order to make (5,5) a punishment value that both players will participate in. The biggest such continuation promise we can make to both players is (6.5,6.5). Hence, the smallest value of δ that supports the (5,5) punishment is $\delta(6.5-3) = 2$ or $\delta = 4/7$. The

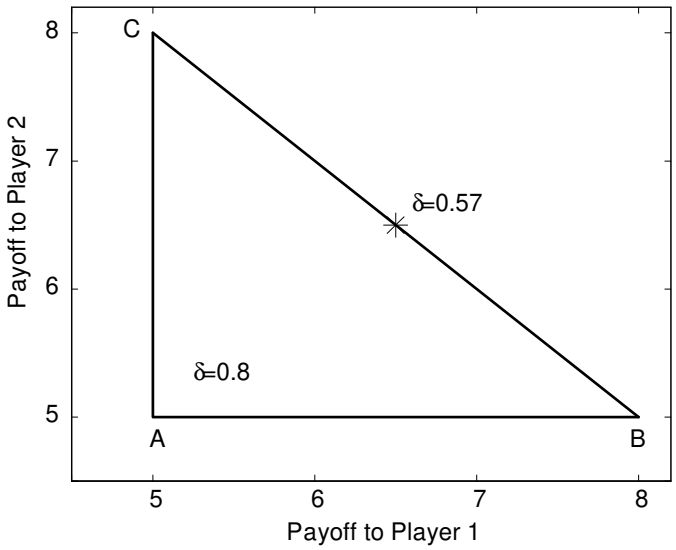


Figure 7: Battle of the Sexes with different δ values.

monotonicity of $B(W)$ in δ tells us that V gets smaller as the discount rate δ decreases. In this example the monotonicity is rather abrupt: the equilibrium value set is defined by the triangle ABC in Figure 7 for $\delta \in (4/7, 1)$, but is reduced to the line segment BC for $\delta \in (0, 4/7)$. Again, our algorithm was able to correctly compute the V for values of δ, even ones close to the critical value $\delta = 4/7$.

5.3. Bertrand Duopoly. We next examine a Bertrand duopoly game with imperfect substitutes. The demand functions are

$$\begin{aligned} q_1 &= \max\{1 - ap_1 + bp_2,\ 0\} \\ q_2 &= \max\{1 - ap_2 + bp_1,\ 0\} \end{aligned}$$

where q_i is firm i's sales and p_i is firm i's price, $i = 1, 2$. Each firm has per-unit costs of c and one-period profits are $\Pi_i(p_1, p_2) = (p_i - c)q_i$. The static Bertrand solution is $p_{\text{Bert}} = (1 + ac)/(2a - b)$ and the collusive solution is $p_{\text{coll}} = ((b - a)c - 1)/(2(b - a))$. If $a = 0.5$, $b = 0.25$, and $c = 0.1$ then $p_{\text{Bertand}} = 1.40$ and $p_{\text{collusion}} = 2.05$. The Bertrand game usually involves continuous choices for prices, but our algorithm allows only a finite number of possible actions. This Bertrand example shows how to apply our algorithm to approximate solutions to problems with continuous strategic choices. We assume that

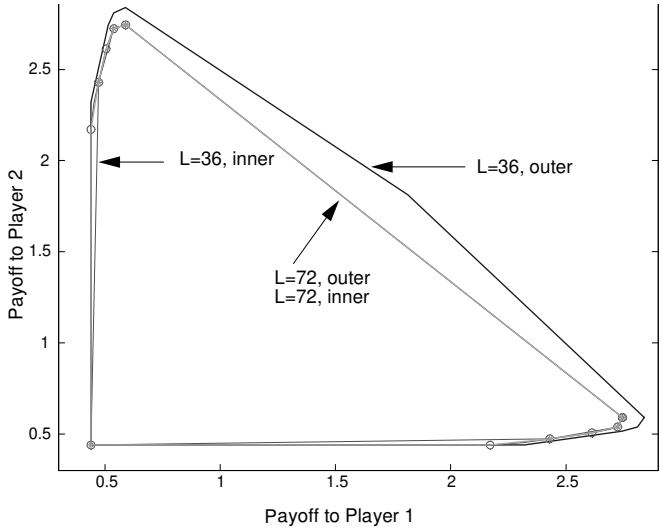


Figure 8: Bertrand Duopoly: Outer and Inner Approximations

the firms may choose among twelve prices, uniformly chosen between 0 to 6.6. The value set appears in Figure 8. The payoffs from price competition are never zero in equilibrium since players can always generate some profit. Hence, their minmax payoffs are positive and the lower bound on V in both dimensions is positive.

We next show the importance of doing both inner and outer approximations. Figure 8 displays the inner and outer solutions to Algorithm 6 when we use L subgradients for $L = 36, 72$. Note that there is a substantial region between the inner and outer approximations when 36 gradients are used, but that there is virtually no difference between the inner and outer approximations when we use 72 gradients. A more precise examination shows that the maximum difference between the two sets in the 72-gradient example is 0.8×10^{-3}, or 0.16% of the players' minmax payoffs. This is an example of where we cannot rely solely on outer approximation methods even for a simple, standard game. In this example, it was the outer approximation which was bad when we used 36 gradients; it is clear that we could also construct examples where the inner approximation was inferior. In this example the differences became small when $L = 72$ but there is no reason to believe that $L = 72$ will always work. The only way to get a good approximation on the equilibrium value set, or even just the Pareto frontier, is to compute both inner and outer approximations. In particular, this example shows the danger of following Cronshaw (1997) and using

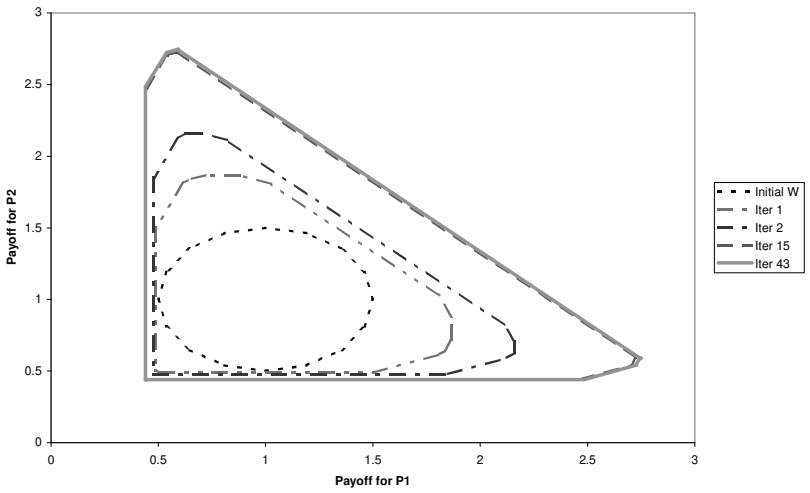


Figure 9: Bertrand Duopoly with a small initial value set.

an outer approximation to compute equilibrium values and strategies since many of the points on the Pareto frontier of the $L = 36$ outer approximation are not close to any true equilibrium value.

Figure 9 displays a different use of inner monotone approximations of $B(W)$. We set the initial guess W^0 equal to a circle of radius 0.5 centered on (1,1). This circle contains the Nash equilibrium of (1.4,1.4) but also many Pareto inferior points which can be used as punishments. We then compute $W^{k+1} = B^I(W^k; H)$ for $k < 44$. We found that $W^0 \subset W^1$. Lemma 8 then tells us that the W^k sequence is monotone increasing and $W^k \subset V$ for all k. The 43'rd iterate displayed in Figure 9 is indistinguishable from the 72 gradient inner approximation displayed in Figure 8. Therefore, we have computed a very good approximation to V just by finding a W^0 which can be proven to be in V by the Lemma 8 criterion, and then iterating B^I several times. We may get lucky, as in this case, and find a W^0 for which this works without too much work. In general, the bounce step of Algorithm 6 does this in a more systematic fashion since it searches for W^0 sets slightly smaller than an outer approximation.

5.4. Cournot Duopoly. We next present solutions to some Cournot duopoly games. This example also shows how to apply our methods to games with a continuous strategy space. We assume q_i is firm i's sales, c_i is firm i's unit cost, p_i is firm i's price, and $\Pi_i(q_1, q_2)$

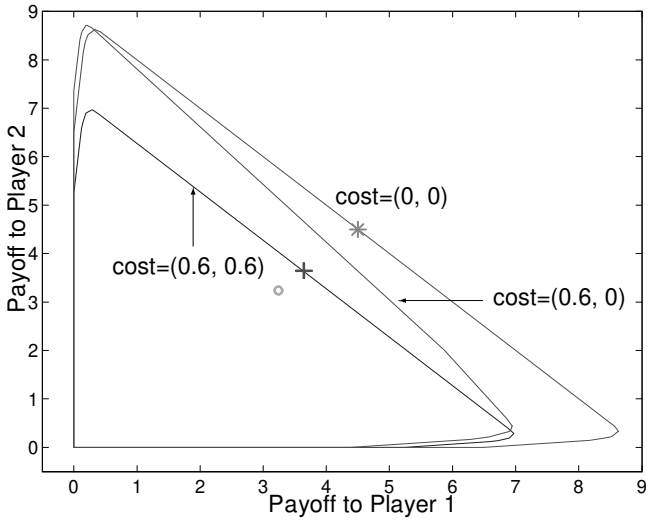


Figure 10: Cournot oligopoly: different costs

is firm i's profits. We assume a linear demand function, $p = \max\{6 - q_1 - q_2,\ 0\}$, and the profit function $\Pi_i(q_1, q_2) = q_i(p - c_i)$. Our approach is limited to finite action games; to accommodate the Cournot duopoly game, we discretize the action space over q_i. As the lumpiness in quantity decisions is reduced, the game resembles its continuous action counterpart. We did runs with 5, 10 and 15 permissible quantities in $[0, 6]$; the solutions we present below used the 15-move specification. We assume the discount rate $\delta = 0.8$.

Figure 10 shows V for cases where firms' costs are either 0.6 or 0. In all cases, the inner and outer approximations are indistinguishable at the resolution of the figures. If both firms have zero costs, a Cournot-Nash equilibrium output is (2, 2) with payoffs equal to (4, 4). Monopoly output is 3, monopoly profits are 9, and the symmetric monopoly payoff, labelled '*' in Figure 10, is (4.5, 4.5). Figure 10 also shows the case where the firms' costs are identical with $c_i = 0.6$, $i = 1, 2$, and the Cournot-Nash output is (1.8, 1.8) with discounted profits (3.24, 3.24) labeled by 'o'. The set of equilibrium values is quite large in both cases. The point (0,0) is a Nash equilibrium value in the zero cost case because playing (6,6) forever is a Nash equilibrium. However, (0,0) is still in V when $c_1 = c_2 = 0.6$ and the unique static Nash equilibrium is (1.8,1.8). In the positive cost case the threats are severe, far worse than Nash-Cournot reversion, and the equilibrium value set is large.

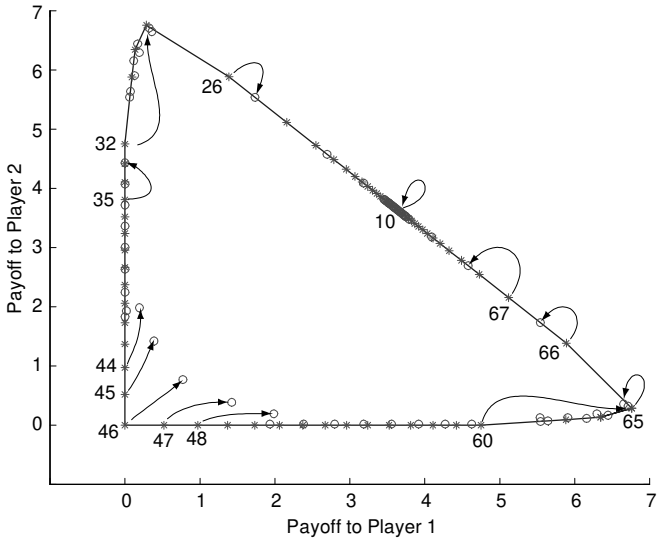


Figure 11: Cournot paths

We use the inner ray step of Algorithm 6 to compute strategies and understand V. Figure 11 and Table 1 describes the actions which support the Cournot game with $c = (0.6, 0.6)$. Figure 11 illustrates the relation between current values and continuation values. The '*' points are at the end of equiangular rays centered on the Nash equilibrium at (3.24, 3.24). Each point ℓ is supported by some action profile and some continuation value where the continuation value, represented by an '$\circ$', is indicated by an arrow beginning at point ℓ. Table 1 displays precise information for 7 points from Figure 11. In Table 1, each ℓ corresponds to the point marked ℓ in Figure 11, $v_i(\ell)$ is player i's equilibrium value at point ℓ, $\bar{v}_i(\ell)$ is his continuation value, q_i is his current output, and $\Pi_i(q_1, q_1)$ is player i's current period payoff.

Figure 11 and Table 1 shows us that points on southern (and western) extreme points of V, the punishment points, usually involve "going along with your own punishment". This reflects results from Abreu (1988), and demonstrates the connection between "simple paths" and the value set approach to repeated games. Players go along with their punishment here by producing so much that one player (and at point 45, both players) make losses in the current period. This is rational for each player because of the promise in the future that he will make higher profits. This logic and the information in Table 1 explains

why (0,0), point 46, is in V. At point 46, both firms produce $q = 5.1$ which produces immediate losses of -3.0 for each, but each firm enjoys an expected continuation value equal to 0.7714. Table 1 also shows that asymmetric harsh punishments yield equally asymmetric continuation values; for example, point 60 gives positive value to firm 1 but zero to firm 2. Table 1 shows that this is implemented by both making negative profits initially but large profits in the future for firm 1 and small future profits for firm 2. Point 60 is also the point which punishes firm 2 if it deviates from cooperation.

Table 1: Actions, promises, and threats on the boundary of V, $c = 0.6$

ℓ	$(v_1(\ell), v_2(\ell))$		$(\bar{v}_1(\ell),\ \bar{v}_2(\ell))$		(q_1, q_2)		$\Pi(q_1, q_2)$	
2	3.97	3.30	3.75	3.52	1.7	0.9	4.8	2.4
8	3.71	3.57	3.72	3.55	1.3	1.3	3.6	3.6
10	3.64	3.64	3.64	3.64	1.3	1.3	3.6	3.6
27	0.29	6.76	0.36	6.65	0.0	3.0	0.0	7.1
46	0.000	0.00	0.77	0.77	5.1	5.1	-3.0	-3.0
60	4.75	0.00	6.71	0.32	5.1	2.1	-3.0	-1.3

Figure 10 also displays the asymmetric cost case with $c_1 = 0.6$ and $c_2 = 0.0$. The asymmetry in costs causes V to be asymmetric. Figure 10 shows that an increase in costs for firm 1 can improve equilibrium payoffs to firm 2 since firm 2's maximum payoff is greater when c_1 increases from zero to 0.6, holding firm 1's costs constant at zero. However, the primary effect is that firm 1's maximum possible payoff is reduced from almost 9 to about 7. Figure 10 displays results like comparative statics – as a firm's costs rises, the set of equilibrium values shifts in favor of the other firm – but the large number of equilibria limits the precision.

5.5. Timing Examples. Our presentation focussed on describing the steps of our algorithm and deriving error bounds for the computed value sets. An algorithm also needs to be practical, running in reasonable time on widely available machines using standard software. We next report some running times for the Cournot game studied in the previous section.

We report the run times for different choices of the number of search gradients and the number of actions per player. We used a compiled Fortran program on a 500 MHz Pentium PC. The run times reflect the total amount of time for all the steps in Algorithm

6, computing an outer approximation, an inner approximation, executing the bounce test, and constructing equilibrium actions and continuation values. The initial set is $\mathcal{W}$. We choose two convergence criterion: $\epsilon = 10^{-5}, 10^{-7}$. These are more demanding than typical in economics, and used here to indicate an upper bound on run time.

Tables 2 and 3 display the running times. More time is typically spent on the first step of computing an outer approximation since that step has a crude initial guess. Since the outer approximation is used to create the initial guess for computing the inner approximation, the inner approximation step takes less time. Table 2 shows how running time is affected by the discretization we use. Increasing the number of possible actions per player increases the number of action profiles which need to be checked. Since we have two players, the action profiles is the square of the number of actions per player. Accordingly, as we increase the number of actions, the running times go up roughly quadratically. The running times go up as we increase the number of search subgradients. Using more subgradients increases the running times for two reasons. First, we check more directions in each iteration. Second, each polytope is approximated using more subgradients, increasing the number of constraints in the $w \in W$ condition.

Table 2: Run times: Actions versus Gradients

$\epsilon = 10^{-5},\ \beta = 0.8$

	Search Gradients		
Actions per player	16	32	72
5	8s	36s	4m 57s
10	28s	1m 34s	17m 7s
15	63s	4m 46s	44m 53s

The choice of stopping criterion also affects running time. Table 3 shows that as we move from $\epsilon = 10^{-7}$ to $\epsilon = 10^{-5}$ running times are cut in about half. This is somewhat surprising since it tells us that just doubling the effort improves accuracy by a factor of 100. Table 3 also indicates how the discount factor can affect running time. As β increases, we expect running times to increase, and they do. Table 3 shows that the increase is moderate, with running times at most doubling when we go from $\beta = 0.8$ to $\beta = 0.9$.

Table 3: Run times: Stopping Criterion and Discount Factors

16 Search Subgradients

Actions per player	$\epsilon = 10^{-7},\ \beta = 0.8$	$\epsilon = 10^{-5},\ \beta = 0.8$	$\epsilon = 10^{-5},\ \beta = 0.9$
5	15s	8s	11s
10	50s	28s	45s
15	1m 59s	63s	1m 54s

6. Conclusion

This paper has described and implemented a computer algorithm to solve discounted supergames with perfect monitoring. Key assumptions are restriction to pure strategies and the inclusion of public randomization. We represent the key object in the dynamic programming approach to supergames, the value set, by its boundary. This parsimonious representation of the value set allows us to compute an approximations of the set along with error bounds, and do so more rapidly than would be possible using "brute force" methods such as exhaustive search.

We demonstrate the algorithm for a variety of games with two players: the prisoner's dilemma, the battle of the sexes, and Cournot and Bertrand competition. Example computations are consistent with the theoretical results of Cronshaw and Luenberger, and Abreu, Pearce and Stacchetti, and show that our algorithm can compute values sets with high accuracy. We also see how large value sets can be, and how the worst punishments (optimal penal codes in the language of Abreu (1988)) can be much worse than Nash reversion. In the Cournot example this is especially clear, since the Cournot-Nash point is in the interior and far away from the worst punishment.

The procedures described here can clearly be extended to a variety of other contexts. For example, the authors have extended it to dynamic games with state variables, such as investment capacity. The themes of outer and inner approximations will be central to any computational attempt to solve supergames which can be expressed using the dynamic programming approach.

References

[1] Abreu, Dilip. "On the Theory of Infinitely Repeated Games with Discounting," *Econometrica* **56** (1988), 383–396.

[2] Abreu, Dilip, David Pearce, and Ennio Stacchetti. "Optimal Cartel Equilibria with Imperfect Monitoring," *Journal of Economic Theory* **39** (1986), 251–269.

[3] Abreu, Dilip, David Pearce, and Ennio Stacchetti. "Toward a Theory of Discounted Repeated Games with Imperfect Monitoring," *Econometrica* **58** (1990), 1041–1063.

[4] Atkeson, Andrew . "International Lending with Moral Hazard and Risk of Repudiation," *Econometrica* **59** (1991): 1069-1089.

[5] de Berg, M., M. van Kreveld, M. Overmars and O. Schwarzkopf . *Computational Geometry: Algorithms and Applications.*, Springer-Verlag, Germany. 1997.

[6] Conklin, James, and Kenneth Judd. "Computing Supergame Equilibria," mimeo, Hoover Institution, 1993.

[7] Cronshaw, Mark. "Algorithms for Finding Repeated Game Equilibria", *Computational Economics* **10** (1997): 139-68

[8] Cronshaw, Mark and David G. Luenberger. "Strongly Symmetric Subgame Perfect Equilibrium in Infinitely Repeated Games with Perfect Monitoring and Discounting," *Games and Economic Behavior* **6** (1994): 220 – 237.

[9] Eaves, B. C. "Homotopies for Computation of Fixed Points", *Mathematical Programming* **3** (1972):1–22.

[10] Goodman, Jacob and Joseph O'Rourke, ed.: *Handbook of Discrete and Computational Geometry.*CRC Press, 1997.

[11] Graham, R. L.. "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set," *Info.Proc. Lett.* **1** (1972), 132-133.

[12] Judd, Kenneth L. *Numerical Methods in Economics*, MIT Press, 1998.

[13] Phelan, Christopher and Ennio Stacchetti. "Subgame-perfect Equilibria in a Ramsey Tax Model." Unpublished paper, Federal Reserve Bank of Minneapolis, 1999.

[14] Preparata, Franco and Michael I. Shamos. *Computational Gemetry: An Introduction.*Springer-Verlag, New York, 1989.

[15] Scarf, H. E. "The approximation of fixed points of a continuous mapping", *SIAM Journal of Applied Mathematics*, **15** (1967): 328-343.

[16] Sleet, Christopher. "New Recursive Methods for Macroeconomic Policy Games." mimeo. Stanford University, 1996.

[17] Sleet, Christopher and Sevin Yeltekin. "On the Computation of Value Correspondences," mimeo, KGSM-MEDS, Northwestern University, 2000.

[18] Sorin, Sylvain. "On repeated games with complete information," *Mathematics of Operations Research* **11** (1986), 147-160.