Closed-form Solutions to a Subclass of Continuous Stochastic Games via Symbolic Dynamic Programming

Abstract

Zero-sum stochastic games provide a formalism to study competitive sequential interactions between two agents with diametrically opposing goals and evolving state. A solution to such games in the case of a discrete state, was presented in (Littman, 1994). The continuous state case, however, remains unsolved. In many instances this requires nonlinear parameterised optimisation, a problem for which closed-form solutions are generally unavailable. We characterise a subclass of continuous stochastic games and present a novel symbolic dynamic programming method which enables these problems to be reduced to a parameterised linear program, for which an exact closed-form solution exists. This solution is empirically applied to compute *exact* solutions to a variety of *continuous state* zero-sum stochastic games.

1 Introduction

Modelling competitive sequential interactions between agents has important applications within economics and decision-making. Stochastic games (Shapley, 1953) provide a convenient framework to model sequential interactions between non-cooperative agents. In zero-sum stochastic games, participating agents have diametrically opposing goals. A reinforcement learning solution to zero-sum stochastic games with discrete states was presented in (Littman, 1994). Closed-form solutions for the continuous state case remain unknown, despite the general importance of this formalism — zero-sum continuous state stochastic games provide a convenient framework with which to model robust sequential optimisation in adversarial settings including many important financial and economic domains such as option valuation on derivative markets.

The difficulty of solving zero-sum continuous state stochastic games arises from the need to calculate a Nash equilibrium for every state, of which there are infinitely many. In this paper we characterise a subclass of continuous state stochastic games for which we can calculate exact solutions for arbitrary horizons via closed-form symbolic dynamic programming (SDP) (Boutilier *et al.*, 2001) in continuous domains (Sanner *et al.*, 2011; Zamani & Sanner, 2012).

We begin by presenting Markov Decision Processes (MDPs) (Howard, 1960) and value iteration (Bellman, 1957), a commonly used dynamic programming solution for MDPs. We then describe both discrete and continuous state zero-sum stochastic games as game-theoretic generalisations of the MDP framework. Following this we show how symbolic dynamic programming can be used to calculate the first known exact solution to a particular subclass of zero-sum continuous stochastic games. We conclude by calculating exact solutions to continuous state matching pennies and a binary option valuation game.

In this paper we make the following key contributions:

- We characterise a subclass of zero-sum continuous stochastic games with restricted reward and transition functions that can be solved exactly via parameterised linear optimisation.
- We provide an algorithm that solves this subclass of stochastic games exactly and optimally using symbolic dynamic programming for arbitrary horizons.

2 Markov Decision Processes

A Markov Decision Process (MDP) (Howard, 1960) is defined by the tuple $\langle S, A, T, R, h, \gamma \rangle$. S and A specify a finite set of states and actions, respectively. T is a transition function $T : S \times A \to S$ which defines the effect of an action on the state. R is the reward function $R : S \times A \to \mathbb{R}$ which encodes the preferences of the agent. The horizon h represents the number of decision steps until termination and the discount factor $\gamma \in [0, 1)$ is used to discount future rewards. In general, an agent's objective is to find a

policy, $\pi : S \rightarrow A$, which maximises the expected sum of discounted rewards over horizon h.

Value iteration (VI) (Bellman, 1957) is a general dynamic programming algorithm used to solve MDPs. VI is based on the set of Bellman equations, which mathematically express the optimal solution of an MDP. They provide a recursive expansion to compute: (1) $V^*(s)$, the expected value of following the optimal policy in state s; and (2) $Q^*(s,a)$, which is the expected quality of taking a in state s, then following the optimal policy. The key idea of the algorithm is to successively approximate $V^*(s)$ and $Q^*(s,a)$ by $V^h(s)$ and $Q^h(s,a)$, respectively, at each horizon h. These two functions satisfy the following recursive relationship:

$$Q^h(s,a) = R(s,a) + \gamma \cdot \sum_{s' \in S} T(s,a,s') \cdot V^{h-1}(s') \quad (1)$$

$$V^h(s) = \max_{a \in A} \left\{ Q^h(s,a) \right\} \quad (2)$$

The algorithm is executed by first initialising $V^0(s)$ to zero or the terminal reward. Then for each h, $V^h(s)$ is calculated from $V^{h-1}(s)$ via Equations (1) and (2), until the intended h-stage-to-go value function is computed. Value iteration converges linearly in the number of iterations to the true values of $Q^*(s,a)$ and $V^*(s)$ (Bertsekas, 1987).

MDPs can be used to model multiagent systems under the assumption that other agents are part of the environment and have fixed behaviour. As a result, they ignore the difference between responsive agents and a passive environment (Hu & Wellman, 1998). In the next section we present a game theoretic framework which generalises MDPs to situations with two or more agents.

3 Zero-sum Discrete Stochastic Games

Discrete state stochastic games (DSGs) are formally defined by the tuple $\langle S, A_1, \ldots, A_n, T, R_1, \ldots, R_n, h, \gamma \rangle$. S is a set of discrete states and A_i is the action set available to agent i. T is a transition function $T : S \times A_1 \times \ldots \times A_n \rightarrow \Delta(S)$, where $\Delta(S)$ is the set of probability distributions over the state space S. The reward function $R_i : S \times A_1 \times \ldots \times A_n \rightarrow \mathbb{R}$ encodes the individual preferences of agent i. The horizon h represents the number of decision steps until termination and the discount factor $\gamma \in [0,1)$ is used to discount future rewards. In general, an agent's objective is to find a policy, $\pi_i : S \rightarrow \sigma_i(A_i)$ which maximises the expected sum of discounted rewards over horizon h. Here $\sigma_i(A_i)$ specifies probability distributions over action set A_i. The optimal policy in a DSG may be stochastic, that is, it may define a mixed strategy for each state.

Zero-sum DSGs are a type of DSG involving two agents with diametrically opposing goals. Under these conditions the reward structure for the game can be represented by a single reward function since an agents reward function is simply the opposite of their opponent's. The objective of each agent is to maximise its expected discounted future rewards in the minimax sense. That is, each agent views its opponent as acting to minimise the agent's reward. Zero-sum DSGs can be solved using a technique analogous to value iteration for MDPs (Littman, 1994). The value function, $V^h(s)$, in this setting can be defined as:

$$V^h(s) = \max_{m_{a_1} \in \sigma_1(A_1)} \min_{m_{a_2} \in \sigma_2(A_2)} \sum_{a_1 \in A_1} \sum_{a_2 \in A_2} Q^h(s,a_1,a_2) \cdot m_{a_1} \cdot m_{a_2} \quad (3)$$

where m_{a_i} is a mixed strategy from $\sigma_i(A_i)$. $Q^h(s,a_1,a_2)$, the quality of taking action a_1 against action a_2 in state s, is given by:

$$Q^h(s,a_1,a_2) = R(s,a_1,a_2) + \gamma \cdot \sum_{s' \in S} T(s,a_1,a_2,s') \cdot V^{h-1}(s'). \quad (4)$$

Equation (3) can be further simplified by noting that since the min operation is "inside" the max, the minimum is achieved for a deterministic action choice. This observation leads to the following form:

$$V^h(s) = \max_{m_{a_1} \in \sigma_1(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q^h(s,a_1,a_2) \cdot m_{a_1}. \quad (5)$$

Together Equations (4) and (5) define a recursive method to calculate the optimal solution to zero-sum DSGs. The policy for the opponent can be calculated by applying symmetric reasoning and the Minimax theorem (Neumann, 1928).

3.1 Solution Techniques

Zero-sum DSGs can be solved via discrete linear optimisation. The value function in Equation (5) can be reformulated as a linear program through the following steps:

1. Define $V^h(s)$ to be the value of the inner minimisation term in Equation (5). This leads to the following linear program for a known state s:

   maximise $V^h(s)$
   subject to

   $$V^h(s) = \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q^h(s,a_1,a_2) \cdot m_{a_1} \quad (6a)$$

   $$\sum_{a_1 \in A_1} m_{a_1} = 1;\ m_{a_1} \geq 0 \qquad \forall a_1 \in A_1$$

2. Replace the equality (=) in constraint (6a) with $\leq$ by observing that the maximisation of $V^h(s)$ effectively

pushes the $\leq$ condition to the = case. This gives:

$$\begin{aligned}
&\text{maximise } V^h(s) \\
&\text{subject to} \\
&V^h(s) \leq \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q^h(s, a_1, a_2) \cdot m_{a_1} \qquad (7a) \\
&\sum_{a_1 \in A_1} m_{a_1} = 1;\ m_{a_1} \geq 0 \qquad \forall a_1 \in A_1
\end{aligned}$$

3. Remove the minimisation operator in constraint (7a) by noting that the minimum of a set is less than or equal to the minimum of all elements in the set. This leads to the final form of the discrete linear optimisation problem:

$$\begin{aligned}
&\text{maximise } V^h(s) \\
&\text{subject to} \\
&V^h(s) \leq \sum_{a_1 \in A_1} Q^h(s, a_1, a_2) \cdot m_{a_1}\ \forall a_2 \in A_2 \\
&\sum_{a_1 \in A_1} m_{a_1} = 1;\ m_{a_1} \geq 0 \qquad \forall a_1 \in A_1
\end{aligned}$$

We can now use existing linear programming solvers to compute the optimal solution to this linear program for each $s \in S$ at a given horizon h.

The linear program used to solve zero-sum DSGs cannot be used once the state is continuous since there are infinitely many states. The key innovation of this paper is in showing that continuous state zero-sum games can still be solved through the use of symbolic dynamic programming.

4 Zero-sum Continuous Stochastic Games

Continuous state stochastic games (CSGs) are formally defined by the tuple $\langle \vec{x}, A_1, \ldots, A_n, T, R_1, \ldots, R_n, h, \gamma \rangle$. In CSGs states are represented by vectors of continuous variables, $\vec{x} = (x_1, \ldots, x_m)$, where $x_i \in \mathbb{R}$. The other components of the tuple are as previously defined for discrete stochastic games in the preceding section.

Zero-sum CSGs impose the same restrictions on the number of agents and the reward structure as their discrete state counterparts. The optimal solution to zero-sum CSGs can be calculated via the following recursive functions:

$$Q^h(\vec{x}, a_1, a_2) = R(\vec{x}, a_1, a_2) \quad + \gamma \cdot \int T(\vec{x}, a_1, a_2, \vec{x}') \cdot V^{h-1}(\vec{x}')\, d\vec{x}' \qquad (8)$$

$$V^h(\vec{x}) = \max_{m_{a_1} \in \sigma(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q^h(\vec{x}, a_1, a_2) \cdot m_{a_1} \qquad (9)$$

We can derive Equation (8) from Equation (4) by replacing the s, s' and the $\sum$ operator with their continuous state counterparts, $\vec{x}$, $\vec{x}'$ and $\int$, respectively.

4.1 Solution Techniques

Zero-sum CSGs can be solved using a technique analogous to that presented in Section 3.1. Namely, the value function in Equation (9) can be reformulated as the following continuous optimisation problem:

$$\begin{aligned}
&\text{maximise } V^h(\vec{x}) \\
&\text{subject to} \\
&V^h(\vec{x}) \leq \sum_{a_1 \in A_1} Q^h(\vec{x}, a_1, a_2) \cdot m_{a_1}\ \forall a_2 \in A_2 \qquad (10a) \\
&\sum_{a_1 \in A_1} m_{a_1} = 1;\ m_{a_1} \geq 0 \qquad \forall a_1 \in A_1
\end{aligned}$$

This optimisation problem cannot be easily solved using existing techniques due to two factors: (1) there are infinitely many states in $\vec{x}$; and (2) constraint (10a) is nonlinear in $\vec{x}$ and m_{a_1} for general representations of $Q^h(\vec{x}, a_1, a_2)$. To further illustration the second limitation consider $Q^h(\vec{x}, a_1, a_2)$ in the form of a linear function in x for some a_1 and a_2:

$$Q^h(\vec{x}, a_1, a_2) = \sum_j c_j \cdot x_j \qquad (11)$$

Substituting Equation (11) into constraint (10a) yields:

$$V^h(\vec{x}) \leq \sum_{a_1 \in A_1} m_{a_1} \sum_j c_j \cdot x_j \qquad \forall a_2 \in A_2. \qquad (12)$$

It is clear from Equation (12) that a linear representation of $Q^h(\vec{x}, a_1, a_2)$ leads to a nonlinear constraint where m_{a_1} must be optimal with respect to the free variable $\vec{x}$ since we need to solve for all states $\vec{x}$. This results in a parameterised nonlinear program, whose optimal solutions are known to be NP-hard (Bennett & Mangasarian, 1993; Petrik & Zilberstein, 2011).

At this point we present the first key insight of this paper: we can transform constraint (10a) from a parameterised nonlinear constraint to a piecewise linear constraint by imposing the following restrictions: (1) restricting the reward function, $R(\vec{x}, a_1, a_2)$, to piecewise constant functions; and (2) restricting the transition function, $T(\vec{x}, a_1, a_2, \vec{x}')$, to piecewise linear functions.

In the next section we show that zero-sum CSGs, with the aforementioned restrictions, can be solved optimally for arbitrary horizons using symbolic dynamic programming.

5 Symbolic Dynamic Programming

Symbolic dynamic programming (SDP) (Boutilier *et al.*, 2001) is the process of performing dynamic programming via symbolic manipulation. In the following sections we present a brief overview of SDP for zero-sum continuous stochastic games. We refer the reader to (Sanner *et al.*,

2011; Zamani & Sanner, 2012) for more thorough expositions of SDP and its operations.

5.1 Case Representation

Symbolic dynamic programming assumes that all symbolic functions can be represented in case statement form (Boutilier *et al.*, 2001) as follows:

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases}$$

Here the ϕ_i are logical formulae defined over the state $\vec{x}$ and can include arbitrary logical combinations of boolean variables and linear inequalities over continuous variables. Each ϕ_i is disjoint from the other ϕ_j ($j \neq i$) and may not exhaustively cover the state space. Hence, f may only be a partial function. In this paper we restrict the f_i to be either linear or constant functions of the state variable $\vec{x}$. We require f to be continuous.

5.2 Case Operations

Unary operations on a case statement f, such as scalar multiplication $c \cdot f$ where $c \in \mathbb{R}$ or negation $-f$, are applied to each f_i ($1 \leq i \leq k$).

Binary operations on two case statements are executed in two stages. Firstly, the cross-product of the logical partitions of each case statement is taken, producing paired partitions. Finally, the binary operation is applied to the resulting paired partitions. The "cross-sum" $\oplus$ operation can be performed on two cases in the following manner:

$$\begin{cases} \phi_1 : f_1 \\ \phi_2 : f_2 \end{cases} \oplus \begin{cases} \psi_1 : g_1 \\ \psi_2 : g_2 \end{cases} = \begin{cases} \phi_1 \wedge \psi_1 : & f_1 + g_1 \\ \phi_1 \wedge \psi_2 : & f_1 + g_2 \\ \phi_2 \wedge \psi_1 : & f_2 + g_1 \\ \phi_2 \wedge \psi_2 : & f_2 + g_2 \end{cases}$$

"cross-subtraction" $\ominus$ and "cross-multiplication" $\otimes$ are defined in a similar manner but with the addition operator replaced by the subtraction and multiplication operators, respectively. Some partitions resulting from case operators may be inconsistent and are thus removed.

Minimisation over cases, known as $\mathrm{casemin}$, is defined as:

$$\mathrm{casemin}\left(\begin{cases} \phi_1 : f_1 \\ \phi_2 : f_2 \end{cases}, \begin{cases} \psi_1 : g_1 \\ \psi_2 : g_2 \end{cases} \right) = \begin{cases} \phi_1 \wedge \psi_1 \wedge f_1 < g_1 : f_1 \\ \phi_1 \wedge \psi_1 \wedge f_1 \geq g_1 : g_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 < g_2 : f_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 \geq g_2 : g_2 \\ \vdots \qquad\qquad\qquad\quad \vdots \end{cases}$$

$\mathrm{casemin}$ preserves the linearity of the constraints and the constant or linear nature of the f_i and g_i. If the f_i or g_i are quadratic then the expressions $f_i > g_i$ or $f_i \leq g_i$ will be at most univariate quadratic and any such constraint can be linearised into a combination of at most two linear inequalities by completing the square.

The $\mathrm{casemax}$ operator, which calculates symbolic case maximisation, is defined as:

$$\mathrm{casemax}\left(\begin{cases} \phi_1 : f_1 \\ \phi_2 : f_2 \end{cases}, \begin{cases} \psi_1 : g_1 \\ \psi_2 : g_2 \end{cases} \right) = \begin{cases} \phi_1 \wedge \psi_1 \wedge f_1 > g_1 : f_1 \\ \phi_1 \wedge \psi_1 \wedge f_1 \leq g_1 : g_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 > g_2 : f_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 \leq g_2 : g_2 \\ \vdots \qquad\qquad\qquad\quad \vdots \end{cases}$$

$\mathrm{casemax}$ preserves the linearity of the constraints and the constant or linear nature of the f_i and g_i.

Other important SDP operations include substitution and continuous maximisation. The substitution operation simply takes a set θ of variables and their substitutions, e.g. $\theta = \left\{x_1'/(x_1 + x_2), x_2'/x_1^2 \exp(x_2)\right\}$, where the LHS of the / represents the substitution variable and the RHS of the / represents the expression that should be substituted in its place. We can apply the substitution θ to both non-case functions f_i and case partitions ϕ_i as $f_i\theta$ and $\phi_i\theta$, respectively. Substitution into case statements can therefore be written as:

$$f\theta = \begin{cases} \phi_1\theta : & f_1\theta \\ \vdots & \vdots \\ \phi_k\theta : & f_k\theta \end{cases}$$

Maximisation can be used to calculate $f_1(\vec{x}, y) = \max_y f_2(\vec{x}, y)$ with respect to a continuous parameter y. This is a complex case operation whose explanation is beyond the scope of this paper. We refer the reader to (Zamani & Sanner, 2012) for further elucidation on this operator.

Case statements and their operations are implemented using Extended Algebraic Decision Diagrams (XADDs) (Sanner *et al.*, 2011). XADDs provide a compact data structure with which to maintain compact forms of $Q^h(\vec{x}, a_1, a_2)$ and $V^h(\vec{x})$. XADDs also permit the use of linear constraint feasibility checkers to prune unreachable paths in the XADD.

5.3 SDP for Zero-sum Continuous Stochastic Games

In this section we show that SDP can be used to calculate optimal closed-form solutions to zero-sum continuous stochastic games with piecewise constant rewards and a piecewise linear transition functions.

We calculate the optimal solution to zero-sum CSGs by first expressing $R(\vec{x}, a_1, a_2)$, $T(\vec{x}, a_1, a_2, \vec{x}')$, $V^0(\vec{x})$ and the m_{a_1} as case statements. We can then compute the optimal solution via the following SDP equations:

$$Q^h(\vec{x}, a_1, a_2) = R(\vec{x}, a_1, a_2) \quad \oplus \\ \gamma \otimes \int T(\vec{x}, a_1, a_2, \vec{x}') \otimes V^{h-1}(\vec{x}')\, d\vec{x}' \quad (13)$$

$$V^h(\vec{x}) = \max_{m_{a_1} \in \sigma(A_1)} \left[\left\{ \text{casemin}_{a_2 \in A_2} \left(\sum_{a_1 \in A_1} Q^h(\vec{x}, a_1, a_2) \otimes m_{a_1} \right) \right\} \otimes \mathbb{I} \right] \quad (14)$$

Equations (13) and (14) can be derived from Equations (8) and (9) by replacing all functions by their case statement equivalents and replacing operations such as $+$, $\times$ and min, by their symbolic equivalents, $\oplus$, $\otimes$ and casemin, respectively. Equation (14) also includes an "indicator" function, $\mathbb{I}$, which serves as a summation constraint $\sum_{a_1 \in A_1} m_{a_1} = 1$. The function $\mathbb{I}$ returns 1 when the conjunction of all constraints on each m_{a_1} are satisfied and $-\infty$, otherwise. That is:

$$\mathbb{I} = \begin{cases} \forall a_1 \in A_1 \quad [(m_{a_1} \geq 0) \wedge (m_{a_1} \leq 1) \wedge (\sum m_{a_1} = 1)] : 1 \\ \forall a_1 \in A_1 \neg [(m_{a_1} \geq 0) \wedge (m_{a_1} \leq 1) \wedge (\sum m_{a_1} = 1)] : -\infty \end{cases}$$

The summation constraint is included to ensure that the subsequent max operation returns legal values for m_{a_1}, since $\max(l, -\infty) = l$.

In Algorithm 1 we present a methodology to calculate the optimal h-stage-to-go value function, $V^h(\vec{x})$, Equations (13) and (14). For the base case of $h = 0$, we set $V^0(\vec{x})$, expressed in case statement form, to zero or to the terminal reward. For all $h > 0$ we apply the previously defined SDP operations in the following steps:

1. Prime the Value Function. In line 6 we set up a substitution $\theta = \{x_1/x'_1, \ldots, x_m/x'_m\}$, and obtain $V'^h = V^h\theta$, the "next state".

2. Discount and add Rewards. We assumed that the reward function contains primed variables incorporate it in line 9.

3. Continuous Marginal Integration. For the continuous state variables in $\vec{x}$ lines 12 - 13 follow the rules of integration w.r.t. a δ function (Sanner *et al.*, 2011) which yields the following symbolic substitution: $\int f(x'_j) \otimes \delta\left[x'_j - g(\vec{z})\right] dx'_j = f(x'_j)\left\{x'_j/g(\vec{z})\right\}$, where $g(\vec{z})$ is a case statement and $\vec{z}$ does not contain x'_j. The latter operation indicates that any occurrence of x'_j in $f(x'_j)$ is symbolically substituted with the case statement $g(\vec{z})$.

4. Incorporate Agent 1's unknown stochastic strategy. At this point we have calculated $Q^h(\vec{x}, a_1, a_2)$, as shown in Equation (13). In lines 16 - 17 we proceed to incorporate each $m_{a_1} \in \sigma_1(A_1)$. We note that each m_{a_1} is a case statement representing a free variable.

5. Case Minimisation. In lines 20 - 21 we minimise $Q^h(\vec{x}, a_1, a_2)$ over all $a_2 \in A_2$ using a sequence of casemin operations.

6. Incorporate Constraints. In line 24 we incorporate constraints on m_{a_1} via the constraint indicator function $\mathbb{I}$. The casemax operator ensures that all case partitions which involve illegal values of m_{a_1} are mapped to $-\infty$.

7. Maximisation. In lines 28 - 29 we maximise over the $m_{a_1} \in \sigma_1(A_1)$ by sequentially applying the maximisation operator as defined previously. At this point we have calculated $V^h(\vec{x})$ as shown in Equation (14).

It can be proved that all SDP operations on case statements result in a linear piecewise constant $V^h(\vec{x})$, given a linear piecewise constant $V^0(\vec{x})$ (Sanner *et al.*, 2011; Zamani & Sanner, 2012).

In the next section we demonstrate how SDP can be used to compute exact solutions to a variety of zero-sum continuous stochastic games.

6 Empirical Results

In this section we evaluate our novel SDP solution technique on three domains: (1) continuous stochastic matching pennies; (2) continuous binary option valuation; and (3) continuous energy production. The domain descriptions and results are presented below.

6.1 Continuous Stochastic Matching Pennies

Matching pennies is a well known zero-sum game with a mixed Nash Equilibrium (Osborne, 2004). In this paper we extend the standard formulation of the game by incorporating continuous state and sequential decisions while still maintaining the zero-sum nature of the reward.

6.1.1 Domain Description

We define continuous stochastic matching pennies as an extensive form game between two players $p \in \{1, 2\}$. The aim of a player is to maximise its expected discounted payoff at a fixed horizon H. Our game is played within the interval $[0, 1]$, two fixed variables $c \in [0, 1)$ and $d \in (0, 1]$ with $(c < d)$, are used to partition the interval into three regions $r \in \{1, 2, 3\}$. Each region is associated with its own zero-sum reward structure. The continuous state variable $x \in [0, 1]$ is used to specify which region the the players are competing within.

Algorithm 1: CSG-VI(CSG, $H, \mathbb{I}$) $\longrightarrow (V^h)$

begin
  $V^0 := 0, h := 0$
  while $h < H$ **do**
    $h := h + 1$
    // Prime the value function
    $Q^h := \texttt{Prime}\,(V^{h-1})$

    if *R contains primed variables* **then**
      $Q^h := R(\vec{x}, a_1, a_2, \vec{x}') \oplus (\gamma \otimes Q^h)$

    // Continuous marginal integration
    for *all* x'_j *in* Q^h **do**
      $Q^h := \int Q^h \otimes T(x'_j | a_1, a_2, \vec{x}) d_{x'_j}$

    // Incorporate Agent 1's unknown stochastic strategy
    for *all* a_1 *in* A_1 **do**
      $Q^h := Q^h \oplus \left(Q^h \otimes m_{a_1}\right)$

    // Case Minimisation
    for *all* a_2 *in* A_2 **do**
      $Q^h := \text{casemin}(Q^h, a_2)$

    // Incorporate constraints
    $Q^h := \text{casemin}(Q^h, \mathbb{I})$

    // Maximisation
    $V^h = Q^h$
    for *all* a_1 *in* A_1 **do**
      $V^h := \max\left(V^h, m_{a_1}\right)$

    // Terminate if early convergence
    if $V^h = V^{h-1}$ **then**
      break
  return (V^h)

At each horizon ($h \leq H$) each player executes an action $a_p \in \{heads_p, tails_p\}$. Player 1 "wins" if both players choose the same action. Otherwise, Player 2 wins. The joint actions of the players affect the state x as follows:

$$P(x'|x, a_1, a_2) =$$

$$\delta\left[x' - \begin{cases} (heads_1) \wedge (heads_2) \wedge (x \geq k): & x - k \\ (heads_1) \wedge (tails_2) \wedge (x \leq 1): & x + k \\ (tails_1) \wedge (heads_2) \wedge (x \geq k): & x + k \\ (tails_1) \wedge (tails_2) \wedge (x \leq 1): & x - k \end{cases}\right]$$

The constant $k \in (0, 1)$ is a step size which perturbs the state x. If Player 1 wins, the state moves to the left by k, otherwise it moves to the right by k. The Dirac function $\delta[\cdot]$ ensures that the transitions are valid conditional probability functions that integrate to 1. We define the rewards obtained by Player 1 in region r as:

$$R_1^r = \begin{cases} (heads_1) \wedge (heads_2): & \alpha_1^r \\ (heads_1) \wedge (tails_2): & \alpha_2^r \\ (tails_1) \wedge (heads_2): & \alpha_3^r \\ (tails_1) \wedge (tails_2): & \alpha_4^r \end{cases} \tag{15}$$

Here we restrict $\alpha_i^r \in \mathbb{R}$. The rewards obtained by Player 2 in the same region are simply $-R_1^r$. Henceforth, we define rewards from the perspective of Player 1. Given this domain description, we investigate the affects of symmetric and asymmetric reward structures on the continuous stochastic matching pennies game. We define a symmetric reward structure as having $\alpha_1^r = \alpha_4^r$ and $\alpha_2^r = \alpha_3^r$. Under a symmetric reward structure the expected reward in each region is the same for both players. An asymmetric reward structure allows the α_i^r to differ in both sign and magnitude. Under this setting the expected rewards for a player may differ across regions. Tables 1 and 2 show examples of symmetric and asymmetric rewards for the continuous stochastic matching pennies domain.

Table 1: Symmetric reward structure for Player 1. Note that symmetric nature of the rewards within each region and the differing rewards between regions.

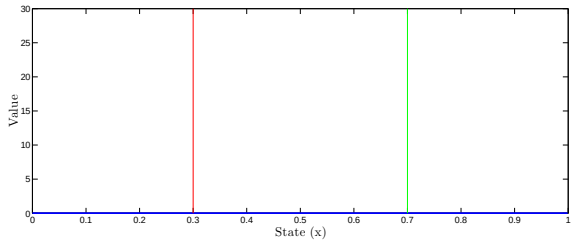
	Region 1	Region 2	Region 3
$(heads_1) \wedge (heads_2)$	10	5	20
$(heads_1) \wedge (tails_2)$	-10	-5	-20
$(tails_1) \wedge (heads_2)$	-10	-5	-20
$(tails_1) \wedge (tails_2)$	10	5	20

Table 2: Asymmetric reward structure for Player 1. Note that asymmetric nature of the rewards within each region and the differing rewards between regions.

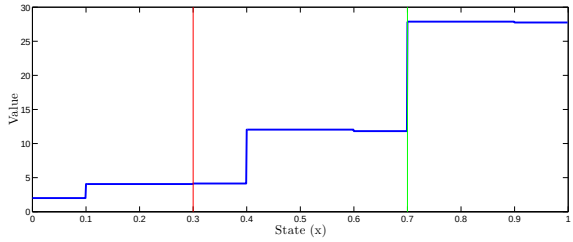
	Region 1	Region 2	Region 3
$(heads_1) \wedge (heads_2)$	1	5	7
$(heads_1) \wedge (tails_2)$	-3	-5	-2
$(tails_1) \wedge (heads_2)$	0	-5	10
$(tails_1) \wedge (tails_2)$	2	5	20

6.1.2 Results

Figure 1a shows the results of the continuous stochastic matching pennies game using the symmetric reward structure given in Table 1. The results show that the expected reward for Player 1 remains at zero over all 4 horizons, irrespective of the state x. Given the symmetric rewards in each region, both players are indifferent between their pure strategies. Hence, the expected reward for each player is zero in all regions. This corresponds to the well known solution of the matching pennies game where the rewards are symmetric and serves as a proof of concept for our novel solution technique.



(a) Symmetric rewards.



(b) Asymmetric rewards.

Figure 1: Optimal value functions for continuous stochastic matching pennies under (a) symmetric and (b) asymmetric reward structures at horizon 4. Threshold values c and d are highlighted in red and green, respectively. The step size is $k = 0.3$.

The effect of the asymmetric reward structure, given in Table 2, is shown in Figure 1b. From the figure it is clear that Player 1 achieves the highest expected reward in Region 3, followed by Region 2 and finally by Region 1. This is to be expected given the nature of the asymmetric rewards within each region. The results indicate that the two players are no longer indifferent between their pure strategies in each region.

6.2 Binary Option Stochastic Game

Binary options are financial instruments which allow an investor to bet on the outcome of a yes/no proposition. The proposition typically relates to whether the price of a particular asset that underlies the option will rise above or fall below a specified amount, known as the strike price, $\kappa \in \mathbb{R}$. When the option reaches maturity the investor receives a fixed pay-off if their bet was correct and nothing otherwise.

6.2.1 Domain Description

We analyse the valuation of a binary option as an extensive form zero-sum game between a trader and the market. The aim of the trader is to maximise their expected discounted pay-off at a fixed horizon H through buying and selling options within an adversarial market. The problem has two state variables: the underlying market value of the option $v \in [0, 100]$ and the trader's inventory of options $i \in \mathbb{N}$.

At each time step the trader can execute one of three actions $a_{trd} \in \{buy_{trd}, sell_{trd}, hold_{trd}\}$, where buy_{trd} refers to a request to buy an option from the market, $sell_{trd}$ refers to a request to sell an option to the market and $hold_{trd}$ is equivalent to taking no action. The market can execute one of two actions: $a_{mkt} \in \{sell_{mkt}, nsell_{mkt}\}$, where $sell_{mkt}$ corresponds to selling an option to the trader and $nsell_{mkt}$ corresponds to not selling to the trader.

The joint actions of the trader and market, a_{trd} and a_{mkt}, respectively, affect both the market value of the option and the trader's inventory. For the sake of simplicity we assume that the market value may increase or decrease by fixed step sizes, $u \in \mathbb{R}$ for an increase and $d \in \mathbb{R}$ for a decrease.

The trader's option inventory dynamics are given by:

$$P(i'|v, i, a_{\text{trd}}, a_{\text{mkt}}) = \delta \left[i' - \begin{cases} (buy_{trd}) \wedge (sell_{mkt}) : & i+1 \\ (sell_{trd}) \wedge (i > 0) : & i-1 \\ otherwise : & i \end{cases} \right]$$

It should be noted that under this formulation the market will always buy an option from the trader when the trader selects $sell_{trd}$. The market value changes according to:

$$P(v'|v, i, a_{\text{trd}}, a_{\text{mkt}}) = \delta \left[v' - \begin{cases} (buy_{trd}) \wedge (sell_{mkt}) : & v+u \\ (sell_{trd}) \wedge (i > 0) : & v-d \\ otherwise : & v \end{cases} \right]$$

Assuming that the strike price $\kappa \in [0, 100]$, the rewards obtained by the trader are given by:

$$R_{trader} = \begin{cases} (sell_{trd}) \wedge (i > 0) \wedge (v > \kappa) : & 1 \\ otherwise : & 0 \end{cases}$$

The market's reward is simply the additive inverse of the trader's reward. Hence, the binary option game is zero-sum.

6.2.2 Results

In Figure 2 we show the optimal value function for the binary option game at horizon 20. The strike price $\kappa = 45.0$ and the increment and decrement values, u and d are both set to 1.0. The value function clearly shows that under this formulation the trader achieves the most reward by selling the option as soon $v > \kappa$. Selling an option triggers the underlying value to decrease, which triggers the trader to buy once the value falls beneath the strike price. This leads to the continual cycling of buying and selling of the option at values close to the strike price κ. In essence the trader behaves like a market maker in that they take both sides of the transaction at values near κ.

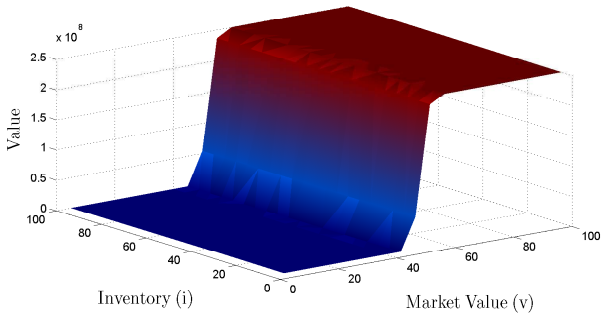


Figure 2: The optimal value function for the binary option game at horizon 20. The strike price is set to $\kappa = 45.0$ and the increment and decrement values are set to $u = 1.0$ and $d = 1.0$, respectively.

6.3 Adversarial Energy Production

The provision of energy resources is an integral component of any economy. Energy providers must be able to produce energy in response to changes in energy demand. In situations where demand exceeds supply, an energy crises may occur. In this paper we investigate energy production from the viewpoint of an energy provider responsible for supplying energy in an adversarial environment.

6.3.1 Domain Description

We define our energy production domain as an extensive form zero-sum game between an energy provider and nature. The aim of the energy provider is to maximise its expected discounted reward at a fixed horizon H by changing production levels in response to changes in demand. The domain has two state variables: the production level $p \in \mathbb{R}$ and the energy demand $d \in \mathbb{R}$.

At each time step the energy provider can execute one of two actions $a_{prd} \in \{dec_{prd}, inc_{prd}\}$, where dec_{prd} refers to increasing energy production and inc_{prd} refers to decreasing energy production. Nature can also execute one of two actions $a_{nat} \in \{dec_{nat}, inc_{nat}\}$, where dec_{nat} refers to increasing energy demand and inc_{nat} refers to decreasing energy demand. For $i \in \{prd, nat\}$ we restrict $\{dec_i, inc_i\} \in \mathbb{Z}$.

The joint actions of the energy provider and nature, a_{prd} and a_{mkt}, respectively, affect the production level as follows:

$$P(p'|d, p, a_{\text{prd}}, a_{\text{nat}}) =$$

$$\delta \left[p' - \begin{cases} (inc_{prd}) : & p + inc_{prd} \\ (dec_{prd}) \wedge (p > dec_{prd}) : & p - dec_{prd} \\ otherwise : & p \end{cases} \right]$$

The energy demand changes according to:

$$P(d'|d, p, a_{\text{prd}}, a_{\text{nat}}) =$$

$$\delta \left[d' - \begin{cases} (inc_{nat}) : & d + inc_{nat} \\ (dec_{nat}) \wedge (d > dec_{nat}) : & d - dec_{nat} \\ otherwise : & d \end{cases} \right]$$

The reward obtained by the energy provider are specified as:

$$R_{prd} = \begin{cases} (p < d) : & -100 \\ otherwise : & 0 \end{cases}$$

We note that under this reward structure not meeting energy demand is heavily penalised, whereas meeting or even exceeding demand are given the same reward. Nature's reward is simply the additive inverse of the energy provider's reward.

6.3.2 Results

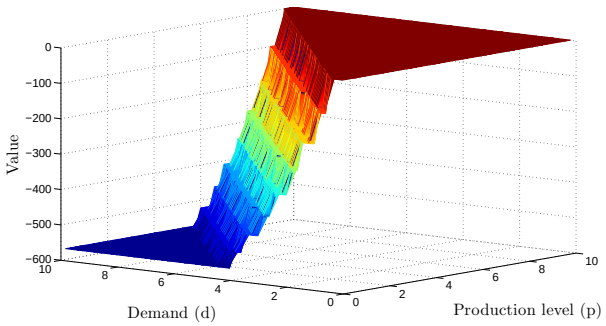


Figure 3: The optimal value function for the adversarial energy production game at horizon 8. The increase and decrease variables where set to $dec_{prd} = inc_{prd} = 1.0$ and $dec_{nat} = inc_{nat} = 0.5$, respectively.

In Figure 3 we show the optimal value function for the adversarial energy production game at horizon 8. The value function shows that the highest value is attained when the energy provided exceeds demand. This is clearly evident given the nature of the reward structure. When the demand exceeds the amount produced, the value is at its lowest. Here we note that the value function decreases in a stepwise manner from the point where where production level meets demand. This indicates that production levels just beneath demand have a higher value than those well below demand.

7 Related Work

Solutions to stochastic games have been proposed from within both game theory and reinforcement learning. The first algorithm, game theoretic or otherwise, for finding a solution to a stochastic game was given by Shapley (Shapley, 1953). The algorithm repeatedly calculates a value function $V(s)$ over discrete states which converges to an optimal value function $V^*(s)$, which represents the expected discounted future reward if both players in the game followed the game's Nash equilibrium. Shapley's algorithm is in essence an extension of the value iteration algorithm to stochastic games. A reinforcement learning based solution to stochastic games was first introduced by Littman (Littman, 1994). Littman's algorithm, Minimax-Q, extends the traditional Q-learning algorithm for MDPs to zero-sum discrete stochastic games. The algorithm converges to the stochastic game's equilibrium solution. Hu and Wellman (Hu & Wellman, 1998) extended Minimax-Q to general-sum games and proved that it converges to a Nash equilibrium under certain restrictive conditions. Although both reinforcement learning based algorithms are able to calculate equilibrium solutions they are limited to discrete state formulations of stochastic games. In this paper we calculate exact solutions to continuous state formulations of stochastic games, under certain restrictions. The Dec-MDP (Bernstein *et al.*, 2002) framework allows for decentralised control within continuous state spaces but is limited to general-sum systems. In this paper we provide the first known exact closed-form solution to a subclass of continuous state zero-sum stochastic games defined by a piecewise constant reward and piecewise linear transition.

Several techniques have been put forward to tackle continuous state spaces in MDPs. Li and Littman (Li & Littman, 2005) describe a method to approximate intermediate piecewise linear value functions by piecewise constant functions, which results in approximate solutions to continuous state MDPs. In this paper we use symbolic dynamic programming to calculate exact solutions to game theoretic domains with continuous state.

Symbolic dynamic programming techniques have been previously used to calculate exact solutions to single agent MDPs with both continuous state and actions in a variety of non-game theoretic domains (Sanner *et al.*, 2011; Zamani & Sanner, 2012). In this paper we present the first application of SDP to game-theoretic domains.

8 Conclusions

We have characterised a subclass of zero-sum continuous stochastic games that can be solved exactly via parameterised linear optimisation. We have also presented a novel symbolic dynamic programming algorithm that can be used to calculated exact solutions to this subclass of stochastic games for arbitrary horizons. The algorithm was used to calculate the first known exact solutions to a variety of experimental continuous domains.

There are a number of avenues for future research. Firstly, it is important to examine more general representations of the reward and transition functions while still guaranteeing exact solutions. Another direction of research lies within improving the scalability of the algorithm by using techniques such as bounded error compression for XADDs (Vianna *et al.*, 2013) or lazy approximation of value functions as piecewise linear XADDs (Li & Littman, 2005). Search based approaches such as RTDP (Barto *et al.*, 1995) and HAO* (Meuleau *et al.*, 2009) are also readily adaptable to SDP. These extensions may then be used to model more complex financial and economic domains. Finally, SDP can be used to calculate exact solutions to general sum games. The advances made by this work open up a number of potential novel research paths that we believe may help make progress in solving game theoretic domains with continuous state.

*References

Barto, Andrew G., Bradtke, Steven J., & Singh, Satinder P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence*, **72**(1-2), 81–138.

Bellman, Richard E. 1957. *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bennett, Kristin P., & Mangasarian, O. L. 1993. Bilinear Separation of Two Sets in N-space. *Comput. Optim. Appl.*, **2**(3), 207–227.

Bernstein, Daniel S., Givan, Robert, Immerman, Neil, & Zilberstein, Shlomo. 2002. The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research*, **27**(4), 819–840.

Bertsekas, Dimitri P. 1987. *Dynamic Programming: Deterministic and Stochastic Models*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Boutilier, Craig, Reiter, Ray, & Price, Bob. 2001. Symbolic Dynamic Programming for First-order MDPs. *Pages 690 – 697 of: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*. IJCAI, vol. 1.

Howard, Ronald A. 1960. *Dynamic Programming and Markov Processes*. The M.I.T. Press.

Hu, Junling, & Wellman, Michael P. 1998. Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm. *Pages 242–250 of: Proceedings of the Fifteenth International Conference on Machine Learning*. ICML. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Li, Lihong, & Littman, Michael L. 2005. Lazy Approximation for Solving Continuous Finite-horizon MDPs. *Pages 1175–1180 of: Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*. AAAI. AAAI Press.

Littman, Michael L. 1994. Markov Games as a Framework for Multi-Agent Reinforcement Learning. *Pages 157–163 of: Proceedings of the 11th International Conference on Machine Learning Machine Learning*. ICML. San Francisco, California, USA: Morgan Kaufmann Publishers Inc.

Meuleau, Nicolas, Benazera, Emmanuel, Brafman, Ronen I., Hansen, Eric A., & Mausam. 2009. A Heuristic Search Approach to Planning with Continuous Resources in Stochastic Domains. *Journal of Artificial Intelligence Research*, **34**, 27–59.

Neumann, J. 1928. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, **100**(1), 295–320.

Osborne, Martin J. 2004. *An introduction to game theory*. New York: Oxford University Press.

Petrik, Marek, & Zilberstein, Shlomo. 2011. Robust Approximate Bilinear Programming for Value Function Approximation. *Journal of Machine Learning Research*, **12**, 3027–3063.

Sanner, Scott, Delgado, Karina, & Nunes de Barros, Leliane. 2011. Symbolic Dynamic Programming for Discrete and Continuous State MDPs. *Pages 1–10 of: Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI-11)*. UAI.

Shapley, L. S. 1953. Stochastic Games. *Proceedings of the National Academy of Sciences*, **39**(10), 1095–1100.

Vianna, Luis Gustavo Rocha, Sanner, Scott, & Nunes de Barros, Leliane. 2013. Bounded Approximate Symbolic Dynamic Programming for Hybrid MDPs. *Pages 1–9 of: Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI2013)*. UAI.

Zamani, Zahra, & Sanner, Scott. 2012. Symbolic Dynamic Programming for Continuous State and Action MDPs. *Pages 1–7 of: Proceedings of the 26th Conference on Artificial Intelligence (AAAI-12)*. AAAI.