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

Shamin Kinathil
ANU and NICTA
Canberra, ACT, Australia
shamin.kinathil@anu.edu.au

Scott Sanner
NICTA and ANU
Canberra, ACT, Australia
ssanner@nicta.com.au

Nicolás Della Penna
ANU and NICTA
Canberra, ACT, Australia
nicolas.della-penna@anu.edu.au

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 with discrete state was presented by Littman (Littman, 1994). The continuous state version of this game remains unsolved. In many instances continuous state solutions require non-linear optimisation, a problem for which closed-form solutions are generally unavailable. We present an exact closed-form solution to a subclass of zero-sum continuous stochastic games that can be solved as a parameterised linear program by utilising symbolic dynamic programming. This novel technique is applied to calculate exact solutions to a variety of zero-sum continuous state stochastic games.

1 INTRODUCTION

Modelling competitive sequential interactions between agents has important applications within economic and financial 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 discrete state zero-sum stochastic games was presented by Littman (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 domains such as option valuation on derivative markets.

The difficulty of solving zero-sum continuous state stochastic games originates from the need to calculate a Nash equilibrium for every state, of which there are infinitely many. In this paper we make the following key contributions:

- We characterise a subclass of zero-sum continuous state 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 (SDP) (Boutilier *et al.*, 2001; Sanner *et al.*, 2011; Zamani & Sanner, 2012) for arbitrary horizons.

This paper is organised as follows: In Section 2 we describe Markov Decision Processes (MDPs) (Howard, 1960) and value iteration (Bellman, 1957), a widely used dynamic programming method for solving MDPs. In Sections 3 and 4, we present zero-sum stochastic games with discrete and continuous states, respectively, as game-theoretic generalisations of the MDP framework. Following this, in Section 5, we introduce SDP, and show how it can be used to calculate the first known exact solution to a particular subclass of zero-sum continuous state stochastic games. In Section 6 we calculate exact solutions to three empirical domains: a continuous state generalisation of matching pennies, binary option valuation and robust energy production. In Section 7, we survey the related literature. We conclude in Section 8 and identify interesting directions for future research.

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 the transition function $T : S \times A \rightarrow S$, which defines the effect of an action on the state. R is the reward function $R : S \times A \rightarrow \mathbb{R}$, 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, $\pi : S \to A$, 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) $Q^*(s, a)$, the expected quality of taking a in state s, then following the optimal policy. The key idea of VI 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 can be 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 by assuming 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 two sections we present game theoretic frameworks which generalises MDPs to situations with two or more responsive 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 $T : S \times A_1 \times \ldots \times A_n \to \Delta(S)$, where $\Delta(S)$ is the set of probability distributions over the state space S. The reward function $R_i : S \times A_1 \times \ldots \times A_n \to \mathbb{R}$, 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, $\pi_i : S \to \sigma_i(A_i)$ 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:

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

where $m \in \mathbb{R}^{|A_1|}$ and $o \in \mathbb{R}^{|A_2|}$ are mixed (stochastic) strategies from $\sigma_1(A_1)$ and $\sigma_2(A_2)$, respectively. $Q^h(s, a_1, a_2)$, the quality of taking action a_1 against action a_2 in state s, is given by:

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

Equation (3) can be further simplified by noting that given any m, the optimal minimum strategy is achieved through a deterministic action choice. This observation leads to the following form:

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

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 at each horizon h. 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 with continuous state formulations, since there are infinitely many states. A key contribution of this paper is in showing that zero-sum continuous state stochastic games can still be solved exactly through the use of symbolic dynamic programming. In the next section we present the continuous state analogue to zero-sum DSGs.

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 in Section 3.

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:

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

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

We can derive Equation (8) from Equation (4) by replacing s, s' and the $\sum$ operator with their continuous state counterparts, $\vec{x}$, $\vec{x}'$ and $\int$, respectively. Equation (9) is simply Equation (5) restated.

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 illustrate 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 $\vec{x}$. 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. As a result, $V^h(\vec{x})$ and $Q^h(\vec{x}, a_1, a_2)$ will be piecewise constant functions, thereby guaranteeing a tractable solution to constraint (10a).

One key challenge still remains, namely, dealing with the infinitely many states in $\vec{x}$. We know that the $V^h(\vec{x})$ and $Q^h(\vec{x}, a_1, a_2)$ functions have structure, but are unable to derive them. Furthermore, given known structures for $V^h(\vec{x})$ and $Q^h(\vec{x}, a_1, a_2)$ we must determine the restrictions that

guarantee a tractable solution. The SDP framework in conjunction with its closed-form operations provide answers to both of these concerns.

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 operations and also show how SDP can be used to solve zero-sum CSGs.

5.1 CASE REPRESENTATION

SDP assumes that all 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 $\vec{x}$ that can consist of arbitrary logical combinations of boolean variables and linear inequalities $(\geq, >, <, \leq)$ over continuous variables. We assume that the set of conditions $\{\phi_1, \ldots, \phi_k\}$ disjointly and exhaustively partition $\vec{x}$ such that f is well-defined for all $\vec{x}$. In general, the f_i may be polynomials of $\vec{x}$ with non-negative exponents. However, in this paper we restrict the f_i to be either constant or linear functions of the state variable $\vec{x}$. Henceforth, we refer to functions with linear ϕ_i and piecewise constant f_i as linear piecewise constant (LPWC) and functions with linear ϕ_i and piecewise linear f_i as linear piecewise linear (LPWL) functions.

5.2 CASE OPERATIONS

Operations on case statements may be either unary or binary. In this section we present a brief overview of the SDP operations needed to calculate closed form solutions to zero-sum CSGs. All of the operations presented here are closed form for LPWC and LPWL functions. We refer the reader to (Sanner *et al.*, 2011; Zamani & Sanner, 2012) for more thorough expositions of SDP and its operations.

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

Binary operations such as addition, subtraction and multiplication 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.

Substitution into case statements is performed via a set θ of variables and their substitutions e.g. $\theta = \{x_1'/(x_1 + x_2), x_2'/x_1^2 \exp(x_2)\}$, where the LHS of the / represents the substitution variable and the RHS of the / represents the expression that should be substituted in its place. θ can be applied 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 be written as:

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

Substitution is used when calculating integrals with respect to deterministic δ transitions (Sanner *et al.*, 2011).

A case statement can be maximised with respect to a continuous parameter y as $f_1(\vec{x}, y) = \max_y f_2(\vec{x}, y)$. The continuous maximisation operation is a complex case operation whose explanation is beyond the scope of this paper. We refer the reader to (Zamani & Sanner, 2012) for further details.

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 $V^h(\vec{x})$.

5.3 SDP FOR ZERO-SUM CONTINUOUS STOCHASTIC GAMES

In this section we will show that a subclass of zero-sum continuous stochastic games with (a) piecewise constant rewards; and (b) piecewise linear transition functions can be solved exactly and in closed-form by using SDP.

To calculate the exact solution to zero-sum CSGs we begin by replacing all functions and operations in Equations (8) and (9) by their case statement equivalents. That is, we exchange operations such as $+$, $\times$ and min, by their symbolic equivalents, $\oplus$, $\otimes$ and $\mathrm{casemin}$, respectively, and express $R(\vec{x}, a_1, a_2)$, $T(\vec{x}, a_1, a_2, \vec{x}')$, $V^0(\vec{x})$ as case statements. m_{a_1} is also encoded as a trivial case statement representing an uninstantiated symbolic variable:

$$m_{a_1} = \Big\{ \top : \quad m_{a_1}$$

The optimal solution to zero-sum CSGs can now be described by the following recursive SDP equations:

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

$$\tilde{Q}^h(\vec{x}, a_2) = \sum_{a_1 \in A_1} Q^h(\vec{x}, a_1, a_2) \otimes m_{a_1} \quad (14)$$

$$V^h(\vec{x}) = \\ \max_m \mathrm{casemin}\left(\mathrm{casemin}_{a_2 \in A_2}\left(\tilde{Q}^h(\vec{x}, a_2)\right), \mathbb{I}\right) \quad (15)$$

Equation (14) calculates a symbolic Q function weighted by the variable m_{a_1} for each a_1. In Equation (15) the inner $\mathrm{casemin}$ operation is calculated with respect to $\tilde{Q}^h(\vec{x}, a_2)$ instantiated with a particular a_2. The "indicator" function $\mathbb{I}$ serves as the summation constraint $\sum_{a_1 \in A_1} m_{a_1} = 1$ and ensures that the subsequent $\max$ operation returns legal values for the m_{a_1}. The indicator is defined as follows:

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

The function $\mathbb{I}$ returns $+\infty$ when the conjunction of all constraints on each m_{a_1} are satisfied and $-\infty$, otherwise.

In Algorithm 1 we present `CSG-VI`, a methodology to calculate the optimal h-stage-to-go value function through Equations (13) to (15). In the algorithm we notationally specify the arguments of the Q^h and V^h functions when they are instantiated by an operation. 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 assume that the reward function contains primed variables and incorporate it in line 8.
3. Continuous Regression. For the continuous state variables in $\vec{x}$ lines 10 – 11 follow the rules of integration w.r.t. a δ function (Sanner *et al.*, 2011). This 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 strategy. At this point we have calculated $Q^h(\vec{x}, a_1, a_2)$, as shown in Equation (13). In lines 13 - 14 we weight the strategy for a specific a_1 by m_{a_1}. We note that m_{a_1} is a case statement representing an uninstantiated symbolic variable.
5. Case Minimisation. In lines 16 – 17 we compute the best case for a_2 as a function of all other variables, as shown in Equation (14).
6. Incorporate Constraints. In line 19 we incorporate constraints on the m_{a_1} variable: $\sum_{a_1 \in A_1} m_{a_1} = 1$ and $m_{a_1} \geq 0 \wedge m_{a_1} \leq 1 \quad \forall a_1 \in A_1$. The $\mathrm{casemin}$ operator ensures that all case partitions which involve illegal values of m_{a_1} are mapped to $-\infty$.
7. Maximisation. In lines 22 – 23 we compute the best response strategy for every state. We note that this can only be done via symbolic methods since there are infinitely many states. At this point we have calculated $V^h(\vec{x})$ as shown in Equation (15).

It can be proved that all of the SDP operations used in Algorithm 1 are closed form for LPWC or LPWL functions (Sanner *et al.*, 2011; Zamani & Sanner, 2012). Given a LPWC $V^0(\vec{x})$ and that SDP operations are closed form, the resulting $V^{h+1}(\vec{x})$ is also LPWC. Therefore, by induction $V^{h+1}(\vec{x})$ is LPWC for all horizons h.

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 for zero-sum CSGs on three continuous domains[1]:

[1]All of the source code can be found online at `http://code.google.com/p/xadd-inference`.

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
    $Q^h := \texttt{Prime}\,(V^{h-1})$
    // Discount and add Rewards
    $Q^h := R(\vec{x}, a_1, a_2, \vec{x}') \oplus (\gamma \otimes Q^h)$
    // Continuous Regression
    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 strategy
    for *all* $a_1 \in A_1$ **do**
      $Q^h := Q^h \oplus \left(Q^h\,(a_1) \otimes \{\top : m_{a_1}\right)$
    // Case Minimisation
    for *all* $a_2 \in A_2$ **do**
      $Q^h := \text{casemin}(Q^h, Q^h\,(a_2))$
    // Incorporate constraints
    $Q^h := \text{casemin}(Q^h, \mathbb{I})$
    // Maximisation
    $V^h = Q^h$
    for *all* $a_1 \in A_1$ **do**
      $V^h := \max_{m_{a_1}} V^h\,(m_{a_1})$
    // Terminate if early convergence
    if $V^h = V^{h-1}$ **then**
      break
  return (V^h)

(1) continuous stochastic matching pennies; (2) binary option stochastic game; and (3) robust energy production. The results represent the first known exact solutions to these domains.

6.1 CONTINUOUS STOCHASTIC MATCHING PENNIES

Matching pennies is a well known zero-sum game with a mixed strategy 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 players are competing within.

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:

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

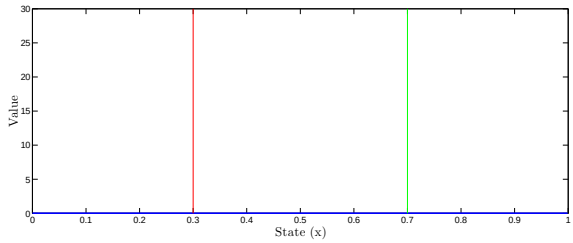
Here we restrict $\alpha_i^r \in \mathbb{R}$. The rewards obtained by Player 2 in the same region are simply $-R_1^r$. Given this reward formulation we specify two different reward structures: symmetric and asymmetric. In a symmetric reward structure $\alpha_1^r = \alpha_4^r$ and $\alpha_2^r = \alpha_3^r$. An example of this reward structure is shown in Table 1. Under a symmetric reward setting the expected reward for each player is the same across all regions *r*. In an asymmetric reward structure we allow each of the α_i^r to differ in both sign and magnitude. Table 2 shows an example of an asymmetric reward structure. Under an asymmetric setting the expected reward for each player may vary across each region *r*. This gives a player an incentive to reach regions with a higher expected reward.

Table 1: Symmetric reward structure for Player 1.

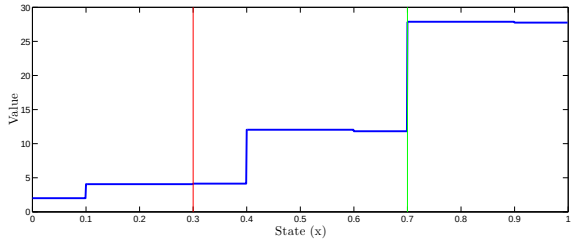
	Region 1	Region 2	Region 3
$(heads_1) \wedge (heads_2)$	10	5	20
$(heads_1) \wedge (tails_2)$	-10	-5	-20
$(tails_1) \wedge (heads_2)$	-10	-5	-20
$(tails_1) \wedge (tails_2)$	10	5	20

Table 2: Asymmetric reward structure for Player 1.

	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



(a) Symmetric rewards.



(b) Asymmetric rewards.

Figure 1: The optimal value functions of the continuous stochastic matching pennies game for Player 1 at horizon 4 under (a) symmetric and (b) asymmetric reward structures. Threshold values are set to $c = 0.3$ and $d = 0.7$ and are highlighted in red and green, respectively. The step size is $k = 0.3$.

6.1.2 Results

We investigate the continuous stochastic matching pennies game under both symmetric and asymmetric rewared structures. For both experiments the threshold values are set to $c = 0.3$ and $d = 0.7$. The step size is $k = 0.3$.

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*. The symmetric reward structure clearly shows that both players achieve the same expected reward in all regions *r*. This in turn ensures that both players are indifferent between their pure strategies. Hence, the expected reward for each player is zero in all regions. This result corresponds to the well known solution of the matching pennies game with symmetric rewards and serves as a proof of concept for our novel solution technique.

Figure (1b) shows the effect of the asymmetric reward structure given in Table 2. The figure shows that Player 1 achieves the highest expected reward in Region 3, followed by Region 2 and finally by Region 1. This corresponds to the expected reward within each region of Table 2. The results indicate that the Player 1 is no longer indifferent between its pure strategies in each region and may take short-term losses to reach more favourable regions.

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 asset $v \in [0, 100]$ 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 underlying market value of the asset 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:

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

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:

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

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

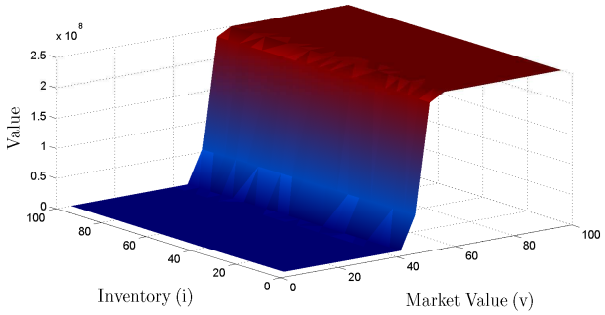


Figure 2: The optimal value function of the binary option stochastic game for the trader at horizon 20. The strike price is set to $\kappa = 45.0$ and the increment and decrement values are set to $u = 1.0$ and $d = 1.0$, respectively. Under the domain specification the value function is invariant to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

In Figure (2) we show 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, 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 options as soon $v > \kappa$. Selling an option causes the underlying value to decrease. Once the value falls beneath the strike price, the trader will buy options, which increases the underlying value. 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. We note that while Figure (2) is invariant to the inventory of options i, its inclusion is critical for the correct formalisation of the domain.

6.3 ROBUST 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 crisis 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 $a_{prd} \in \{inc_{prd}, dec_{prd}\}$, where inc_{prd} refers to increasing energy production and dec_{prd} refers to decreasing energy production. Nature can also execute one of two actions $a_{nat} \in \{inc_{dem}, dec_{dem}\}$, where inc_{dem} refers to increasing energy demand and dec_{dem} refers to decreasing energy demand. We specify the increase in the amount of energy produced or demanded by $prd_u, nat_u \in \mathbb{R}^+$ and a corresponding decrease by $prd_d, nat_d \in \mathbb{R}^+$.

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

$$P(p'|d, p, a_{prd}, a_{nat}) = \delta \left[p' - \begin{cases} (inc_{prd}) : & p + prd_u \\ (dec_{prd}) \wedge (p > prd_d) : & p - prd_d \\ otherwise : & p \end{cases} \right]$$

The energy demand changes according to:

$$P(d'|d, p, a_{prd}, a_{nat}) = \delta \left[d' - \begin{cases} (inc_{dem}) : & d + nat_u \\ (dec_{dem}) \wedge (d > nat_d) : & d - nat_d \\ otherwise : & d \end{cases} \right]$$

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 failure to meet 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

In Figure (3) we show the optimal value function for the robust energy production game at horizon 8. The production and demand increase and decrease variables were set to $prd_u = prd_u = 1.0$ and $nat_u = nat_d = 0.5$, respectively. The value function shows that the energy provider achieves the highest value when the energy provided meets or exceeds demand. The value function is lowest when the demand exceeds supply, which is in accordance with the reward structure. The value function clearly decreases in

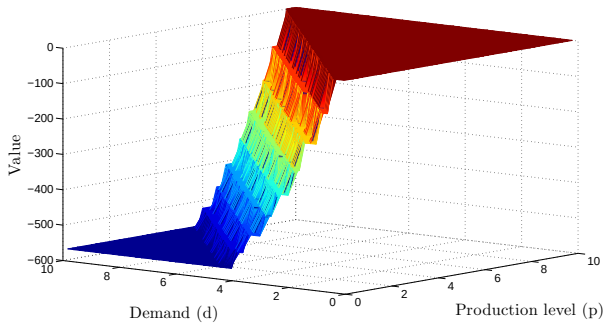


Figure 3: The optimal value function of the robust energy production game for the producer at horizon 8. The production and demand increase and decrease variables were set to $prd_u = prd_u = 1.0$ and $nat_u = nat_d = 0.5$, respectively.

a step-wise manner from the point where the production level meets demand, indicating 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). Shapley's algorithm calculates a value function $V(s)$ over discrete states which converges to an optimal value function $V^*(s)$. $V^*(s)$ represents the expected discounted future reward if both players in the game follow the game's Nash equilibrium. The algorithm is in essence an extension of the Value Iteration algorithm to stochastic games.

A partically implementable solution, based on reinforcement learning, for a subclass of stochastic games was first introduced by (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 provide the first known exact closed-form solution to a subclass of continuous state zero-sum stochastic games defined by piecewise constant reward and piecewise linear transition functions.

Several techniques have been put forward to tackle continuous state spaces in MDPs. Li and Littman (Li & Littman, 2005) describe a method for approximate solutions to continuous state MDPs. In their work, Li and Littman only allow for rectilinearly aligned constraints in their reward and transition functions, not arbitrary linear constraints, and cannot handle general linear transition models without approximation. Our SDP method provides exact solutions without these restrictions, which makes SDP strictly more general. Also, Li and Littman did not consider game-theoretic extensions of their work or the parameterised optimisation problem that these extensions entail.

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 build on this work and present the first application of SDP to stochastic games with concurrently acting agents.

8 CONCLUSIONS

In this paper 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 calculate exact solutions to this subclass of games for arbitrary horizons. The algorithm was used to calculate the first known exact solutions to a variety of continuous stochastic games with piecewise constant reward and piecewise linear transitions.

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 either extending techniques for Algebraic Decision Diagrams (Bahar *et al.*, 1993) from APRICODD (St-Aubin *et al.*, 2000) under the current restrictions on the reward and transition functions, bounded error compression for XADDs (Vianna *et al.*, 2013) for more expressive representations, 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 sequential decision making in more complex financial and economic domains. Finally, SDP can be used to calculate exact solutions to general sum stochastic games. The advances made within this paper open up a number of potential novel research paths which may be used to progress solutions to sequential game theoretic domains with continuous state.

References

Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., & Somenzi, F. 1993. Algebraic Decision Diagrams and Their Applications. *Journal of Formal Methods in Systems Design*, **10**, 171–206.

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.

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-98)*. 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 Twentieth National Conference on Artificial Intelligence (AAAI-05)*. AAAI, vol. 3. AAAI Press.

Littman, Michael L. 1994. Markov Games as a Framework for Multi-Agent Reinforcement Learning. *Pages 157–163 of: Proceedings of the Eleventh International Conference on Machine Learning Machine Learning (ICML-94)*. 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 Twenty-Seventh 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.

St-Aubin, Robert, Hoey, Jesse, & Boutilier, Craig. 2000. APRICODD: Approximate Policy Construction Using Decision Diagrams. *Pages 1089 – 1095 of: Advances in Neural Information Processing 13 (NIPS 2000)*. NIPS. Denver, Colorado, USA: MIT Press.

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 (UAI-13)*. UAI.

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