Best Response Computation in Multiplayer Imperfect-Information Stochastic Games

Sam Ganzfried
Ganzfried Research
Miami Beach, FL
sam.ganzfried@gmail.com

Abstract

Computing a best response is a fundamental task in game theory. One of its uses is to compute the degree of approximation error of an approximation of Nash equilibrium strategies. In many game classes best responses can be computed in polynomial time, but in imperfect-information stochastic games it is equivalent to solving a POMDP and is PSPACE-complete. Prior work has developed an algorithm for computing an approximation of Nash equilibrium strategies in a 4-player imperfect-information naval strategic planning problem which is modeled as a stochastic game. A heuristic approach was developed for computing best responses to determine the approximation error of these strategies, which was shown to have limited scalability. In this paper we present approaches that utilize parallelism to significantly speed up this computation allowing us to compute best responses in significantly larger games.

Introduction

Computing a best response is a fundamental task in games. Typically it can be performed efficiently (in polynomial time), while computing a Nash equilibrium (the standard game-theoretic solution concept) is PPAD-hard for games with more than two players and two-player non-zero-sum games. Since strategies are fixed for the opposing players computing a best response is just a single agent optimization problem, and is often significantly easier than computing solution concepts that involve reasoning about multiple players' strategies. Computing a best response is important for three main reasons. First, if we are able to formulate a prediction for the strategies of the opposing players (e.g., by using an *opponent modeling algorithm*), then computing a best response would allow us to obtain the highest possible expected payoff against this predicted model. Second, several algorithms for computing Nash equilibrium, such as fictitious play (Brown 1951), require the computation of best responses as a subroutine. And third, when an approximation of Nash equilibrium strategies has been computed in a large or complex game, computing a best response for each player allows us to quantify the degree of approximation error (ϵ). The value of ϵ denotes the largest amount that a player can gain by deviating from the strategy profile. In exact Nash equilibrium $\epsilon = 0$, and so naturally our goal is to produce strategies with as small value of ϵ as possible. We say that the computed strategies constitute an ϵ-equilibrium. Formally, for a given candidate (mixed) strategy profile σ^*, define

$$\epsilon(\sigma^*) = \max_{i \in N} \max_{s_i \in S_i} \left[u_i(s_i, \sigma^*_{-i}) - u_i(\sigma^*_i, \sigma^*_{-i}) \right].$$

Here N denotes the set of players, S_i denotes the set of pure strategies for player $i \in N$, and u_i denotes the utility function for player i. σ^*_i is the component of σ^* correspond to player i's strategy, and σ^*_{-i} is the component of σ^* corresponding to the strategies of the players excluding player i.

In a normal-form game (with any number of players) it is clear that a best response computation can be performed in polynomial time. The same is true for extensive-form games of sequential actions with either perfect or imperfect information. While extensive-form game trees can be used to model sequential actions of a known duration (e.g., repeating a simultaneous-move game for a specified number of iterations), they cannot model games of unknown duration, which can potentially contain infinite cycles between states. Such games must be modeled as *stochastic games*.

Definition 1. *A* stochastic game *(aka* Markov game*) is a tuple* (Q, N, A, P, r)*, where:*

- Q *is a finite set of (stage) games (aka* game states*)*
- N *is a finite set of* n *players*
- $A = A_1 \times \ldots \times A_n$*, where* A_i *is a finite set of actions available to player* i
- $P : Q \times A \times Q \rightarrow [0, 1]$ *is the transition probability function;* $P(q, a, \hat{q})$ *is the probability of transitioning from state* q *to state* $\hat{q}$ *after action profile* a
- $R = r_1, \ldots, r_n$*, where* $r_i : Q \times A \rightarrow \mathbb{R}$ *is a real-valued payoff function for player* i

If strategies are fixed for all players excluding player i, then the best response problem for player i in a stochastic game is equivalent to solving a Markov decision process, and can be done in polynomial time. However, if the stochastic game has *imperfect information* (i.e., agents have private information that is not known to other agents), then computing a best response is equivalent to solving a partially observable Markov decision process (POMDP), and is PSPACE-complete. So there is no hope of finding an efficient algorithm for this problem in general; but there may

be specialized algorithms that can efficiently solve specific problems of interest that may have a special structure.

Recently an algorithm has been developed for computing Nash equilibrium strategies in multiplayer stochastic games with imperfect information where the game states form a directed acyclic graph (Ganzfried 2021). This algorithm was used to approximate Nash equilibrium strategies in a 4-player imperfect-information naval strategic planning scenario that was constructed in consultation with a domain expert. In order to evaluate the quality of the computed strategy profile σ^* (i.e., the ϵ), we must compute a best response for each player to σ^*. An algorithm for doing this was developed that recursively calls itself for updated belief states corresponding to new states that can be transitioned to with horizon $t-1$ (where we are interested in a time horizon of t for the initial state). Unfortunately this algorithm was unable to converge in the initial version of the game, which had $K = 300$ non-terminal states, even using several heuristics for improved speed. However, the algorithm was able to solve a smaller version with $K = 150$ with use of these heuristics.

There have been recent applications of game-theoretic algorithms to important problems in national security. These game models and algorithms have differing levels of complexity. Typically these games have two players, and algorithms compute a Stackelberg equilibrium for a model where the "defender" acts as the leader and the "attacker" as the follower; the goal is to compute an optimal mixed strategy to commit to for the defender, assuming that the attacker will play a best response. Computing Stackelberg equilibrium is easier than Nash equilibrium; for example, for two-player normal-form general-sum games, optimal Stackelberg strategies can be computed in polynomial time (Conitzer and Sandholm 2006), while computing Nash equilibrium is PPAD-hard, and widely conjectured that no polynomial-time algorithms exist (Chen and Deng 2006; Daskalakis, Goldberg, and Papadimitriou 2009). Many realistic problems in national security involve more than two agents, sequential actions, imperfect information, probabilistic events, and/or repeated interactions of unknown duration. Several stochastic game models have been previously proposed for national security settings. For example, two-player discounted models of adversarial patrolling have been considered, for which mixed-integer program formulations are solved to compute a Markov stationary Stackelberg equilibrium (Vorobeychik and Singh 2012; Vorobeychik et al. 2014). One work has applied an approach to approximate a correlated equilibrium in a three-player threat prediction game model (Chen et al. 2006). However we are not aware of other prior research on settings with more than two players with guarantees on solution quality (or for computing Nash as opposed to Stackelberg or correlated equilibrium) for either perfect or imperfect information other than the very recent work that we build on (Ganzfried, Laughlin, and Morefield 2020; Ganzfried 2021).

Imperfect-information naval strategic planning problem

We will first review the perfect-information naval strategic planning problem that has been previously studied (Ganzfried, Laughlin, and Morefield 2020), and then describe the differences for the imperfect-information version. The game is based on a freedom of navigation scenario in the South China Sea where a set of *blue* players attempts to navigate freely, while a set of *red* players attempt to obstruct this from occurring (Figure 1). In our model there is a single blue player and several different red players which have different capabilities (we will specifically focus on the setting where there are three different red players). If a blue player and a subset of the red players happen to navigate to the same location, then a confrontation will ensue, which we call a Hostility Game.

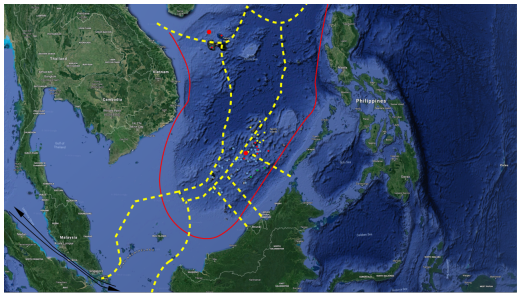

Figure 1: General figure for South China Sea scenario.

In a Hostility Game, each player can initially select from a number of available actions (which is between 7 and 10 for each player). Certain actions for the blue player are *countered* by certain actions of each of the red players, while others are not (Figure 2). Depending on whether the selected actions constitute a counter, there is some probability that the blue player *wins* the confrontation, some probability that the red players win, and some probability that the game repeats. Furthermore, each action of each player has an associated *hostility level*. Initially the game starts in a state of zero hostility, and if it is repeated then the overall hostility level increases by the sum of the hostilities of the selected actions. If the overall hostility level reaches a certain threshold (300), then the game goes into *kinetic mode* and all players achieve a very low payoff (negative 200). If the game ends in a win for the blue player, then the blue player receives a payoff of 100 and the red players receive negative 100 (and vice versa for a red win). The game repeats until either the blue/red players win or the game enters kinetic mode. A subset of the game's actions and parameters are given in Figure 3. Note that in our model we assume that all red players act independently and do not coordinate their actions. The game model and parameters were constructed from discussions with a domain expert.

Definition 2. *A perfect-information hostility game is a tuple* $G = (N, M, c, b^D, b^U, r^D, r^U, \pi, h, K, \pi^K)$*, where*

- N *is the set of players. For our initial model we will assume player 1 is a blue player and players 2–4 are red*

Vessel	Red Move	Blue Counter
Warship	W1	B1,B3,B4,B5,B6,B7,B9
Warship	W2	B2,B9
Warship	W3	B3,B4,B6,B9
Warship	W4	B3,B4,B5,B9
Warship	W5	B1,B3-B7,B9
Warship	W6	B1,B3-B7,B9
Warship	W7	B9
Security Vessel	S1	B1,B3,B4,B5,B6,B7,B8
Security Vessel	S2	B3,B5,B6,B7,B8
Security Vessel	S3	B5,B6,B7,B8
Security Vessel	S4	B5,B6,B7,B8
Security Vessel	S5	B1,B3-B7,B8
Security Vessel	S6	B1,B3-B7,B8
Security Vessel	S7	B8
AUX Vessel	A1	B1,B3,B5,B7,B8
AUX Vessel	A2	B3,B5,B6,B7,B8
AUX Vessel	A3	B5,B7,B8
AUX Vessel	A4	B5,B6,B7,B8
AUX Vessel	A5	B1,B3-B7,B8
AUX Vessel	A6	B1,B3-B7,B8
AUX Vessel	A7	B8

Figure 2: List of blue moves that counter each red move.

players (P2 is a Warship, P3 is a Security ship, and P4 is an Auxiliary vessel).

- $M = \{M_i\}$ *is the set of actions, or* moves*, where* M_i *is the set of moves available to player* i
- *For* $m_i \in M_i$*,* $c(M_i)$ *gives a set of blue moves that are* counter moves *of* m_i
- *For each blue player move and red player, a probability of blue success/red failure given that the move is defended against (i.e.,* countered*), denoted as* b^D
- *Probability that a move is a blue success/red failure given the move is Undefended against, denoted as* b^U
- *Probability for a red success/blue failure given the move is defended against,* r^D
- *Probability for a red success/blue failure given the move is undefended against,* r^U
- *Real valued payoff for success for each player,* π_i
- *Real-valued hostility level for each move* $h(m_i)$
- *Positive real-valued kinetic hostility threshold* K
- *Real-valued payoffs for each player when game goes into Kinetic mode,* π_i^K

We model hostility game G as a (4-player) stochastic game with a collection of stage games $\{G_n\}$, where n corresponds to the cumulative sum of hostility levels of actions played so far. The game has $K+3$ states: $G_0, \ldots, G_K$, with two additional terminal states B and R for blue and red victories. Depending on whether the blue move is countered, there is a probabilistic outcome for whether the blue player or red player (or neither) will outright win. The game will then transition into terminal states B or R with these probabilities, and then will be over with final payoffs. Otherwise, the game transitions into $G_{n'}$ where n' is the new sum of the hostility levels. If the game reaches G_K, the players obtain the kinetic payoff π_i^K. Thus, the game starts at initial state G_0 and after a finite number of time steps will eventually reach one of the terminal states (B, R, G_K).

So far, we have described the Perfect-Information Hostility Game (PIHG). In the real world, often players have some private information that they know but the other players do not. For example, in poker this can be one's private cards, or in an auction one's private valuation for an item. We consider a modification to the PIHG where each player has private information that corresponds to its "strength," or amount of resources it has available. We assume each player has a private *type* t_i from a discrete set T_i, where larger values of t_i correspond to increased strength. We assume that the players know only the value of their own type, while each player knows that the other players' private types are drawn from a public distribution. We assume that each player is drawn a private type t_i from the public distribution at the outset of the game, and that this type persists throughout the game's duration. Thus, the game model of the Imperfect-Information Hostility Game (IIHG) is a 4-player imperfect-information stochastic game.

The values of the type parameters affect the probabilities of each player's success during a confrontation, with larger values leading to greater success probabilities. For example, suppose there is an encounter between a blue ship of type t_b and red ship of type t_r, and suppose that blue player plays an action a_b that is a counter-move to red's action a_r. Then for the PIHG the probability of a blue success would be $p = b^D(a_b)$. In the IIHG we now have that the probability of a blue success will be $p' = p^{\frac{t_r}{t_b}}$. The other success/failure probabilities are computed analogously. Note that for $t_b = t_r$ we have $p' = p$ and the payoffs are the same as for the PIHG. If $t_r > t_b$ then $p' < p$, and similarly if $t_b < t_r$ then $p' > p$.

Prior research for approximating Nash equilibrium strategies in multiplayer imperfect-information stochastic games has focused on the case where the players' private information is *local* and does not extend between game states (Ganzfried and Sandholm 2008; 2009). By contrast, the private information in the IIHG is *persistent* and extends throughout game play. It can also be observed that this problem has the special structure that the game states form a directed acyclic graph. This allowed them to devise a new algorithm that solves each game state sequentially, computes updated type distributions from the state equilibrium strategies at that state, and uses these updated type distributions for solving successive states within the current algorithm iteration. The best algorithm for approximating Nash equilibrium in this game class combines variants of fictitious play (Brown 1951) and policy iteration and is called Sequential Topological FIFP for Type-Dependent Values (ST-PIFP-TDV) (Ganzfried 2021). This was applied to approximate Nash equilibrium strategies in the IIHG.

Prior approach for best response computation

In order to evaluate the strategies computed from our algorithm, we need a procedure to compute the degree of Nash equilibrium approximation, ϵ. For perfect-information stochastic games it turns out that there is a relatively straightforward approach for accomplishing this, based on the observation that the problem of computing a best response for a player is equivalent to solving a Markov decision process (MDP). We can construct and solve a corresponding MDP for each player, and compute the maximum that a player can obtain by deviating from our computed strategies for the initial state G_0. This approach is depicted in Algorithm 1, which applies a standard version of policy iteration (Puterman 2005). It turns out that Algorithm 1 can also be ap-

Move	Vessel	Moves	# of times	Hostility	Probability: Defended	Probability: Undefended
W1	Warship	Bridge to Bridge	1	10	5.00%	7.00%
W2	Warship	Shouldering	Unlimited	25	15.00%	30.00%
W5	Warship	Move towards the Islands	Unlimited	5 + 1 for every lane	5% or 0%	5% or 0%
W6	Warship	Continue on current course	Unlimited	3 + 1 for every lane	5% or 0%	5% or 0%
W7	Warship	K Action	NA	100	45.00%	45.00%
S1	Security Vessel	Bridge to Bridge	1	10	3.00%	5.00%
S2	Security Vessel	Move around AFT end	Unlimited	15	5.00%	15.00%
S4	Security Vessel	Move closer to the ship	Unlimited	55	7.00%	7.00%
S5	Security Vessel	Move towards the Islands	Unlimited	2 + 1 for every lane	3% or 0%	3% or 0%
S6	Security Vessel	Continue on current course	Unlimited	1 + 1 for every lane	3% or 0%	3% or 0%
S7	Security Vessel	K Action	NA	100	25.00%	25.00%
A1	AUX Vessel	Throw out Fishing nets	1	5	3.50%	3.50%
A2	AUX Vessel	Move around AFT end	Unlimited	15	5.00%	5.00%
A3	AUX Vessel	Cross Brow	Unlimited	45	3.00%	3.00%
A4	AUX Vessel	Move closer to the ship	Unlimited	55	7.00%	7.00%
A5	AUX Vessel	Move towards the Islands	Unlimited	2 + 1 for every lane	3% or 0%	3% or 0%
A6	AUX Vessel	Continue on current course	Unlimited	1 + 1 for every lane	3% or 0%	3% or 0%
A7	AUX Vessel	Operates unsafely close to the ship	NA	100	15.00%	15.00%
B1	Blue Ship	Bridge to bridge/Deploy SNOOPIE team	1 For every Vessel interaction	10	W: 5% S&A: 20%	W: 7% S&A: 25%
B2	Blue Ship	Counter Shouldering	Unlimited	30	W: 20% S&A: N/A	N/A
B4	Blue Ship	Continue on current course	Unlimited	2 + 1 for every lane	10.00%	10.00%
B5	Blue Ship	Move towards the Islands	Unlimited	5 + 1 for every lane	0.00%	0.00%
B6	Blue Ship	Speed up	2	15	W: 3% S&A: 25%	W: 3% S&A: 25%
B7	Blue Ship	Alert Crew security	1	60	W: 18% S&A: 75%	W: 18% S&A: 75%
B8	Blue Ship	K Action	NA	100	90.00%	90.00%
B9	Blue Ship	K Action	NA	100	55.00%	55.00%

Figure 3: Sample of typical actions and parameters for Hostility Game.

plied straightforwardly to stochastic games with local imperfect information. This algorithm was applied to compute the degree of Nash equilibrium approximation on the perfect-information hostility game (Ganzfried, Laughlin, and Morefield 2020) and a 3-player imperfect-information poker tournament (Ganzfried and Sandholm 2008; 2009).

Algorithm 1 *Ex post* check procedure

Create MDP M from the strategy profile s^*
Run policy iteration on M (using initial policy s^*) to get π^*
return $\max_{i \in N} \left[v_i^{\pi_i^*, s^*_{-i}}(G_0) - v_i^{s_i^*, s^*_{-i}}(G_0) \right]$

Unfortunately, Algorithm 1 can no longer be applied for stochastic games with persistent imperfect information. For these games, computing the best response for each player is equivalent to solving a partially observable Markov decision processes (POMDP), which is significantly more challenging than solving an MDP. It turns out that computing the optimal policy for a finite-horizon POMDP is PSPACE-complete. The main algorithms are inefficient and typically require an amount of time that is exponential in the problem size (Cassandra, Kaelbling, and Littman 1994). Common approaches involve transforming the initial POMDP to an MDP with continuous (infinite) state space, where each state of the MDP corresponds to a *belief state* of the POMDP.

Due to the problem's intractability, a new procedure was devised for this setting that exploits domain-specific information to find optimal policies in the POMDPs which correspond to computing a best response. The algorithm is based on a recursive procedure, presented in Algorithm 2. The inputs to the procedure are a player i, a type t_i for player i, a set of type *distributions* $\{\tau_j\}$ for the other players $j \neq i$, the strategies computed by our game-solving algorithm $\{s_j^*\}$, a game state G_h, and a time horizon t. The procedure outputs the optimal value in the *belief state* for player i when he has type t_i and the opponents have type distribution $\{\tau_j\}$ at hostility state G_h for the POMDP defined by the strategies $\{s_j^*\}$, assuming that a time horizon of t remains. For simplicity of presentation we assume that $T_i = 2$ for each player (which is what we will use for our experiments), where τ_j denotes the probability that player j has type 1, and $1 - \tau_j$ the probability of type 2. The algorithm recursively calls itself for updated belief states corresponding to new hostility states that can be transitioned to with horizon $t - 1$. As the base case for $t = 0$ we consider only the attainable terminal payoffs from the current state with no additional transitions to new states.

Algorithm 2 ComputeValue$(i, t_i, \{\tau_j\}, \{s_j^*\}, G_h, t)$ (CV)

Inputs: player i, type t_i for player i, type distributions for opposing players $\{\tau_j\}$, strategies for opposing players $\{s_j^*\}$, hostility state G_h, time horizon t

max-payoff $= -\infty$
for each action a_i for player i **do**
  payoff $= 0$
  sum $= 0$
  for every possible combination of $\alpha_k = \prod_j \gamma_j$, where $\gamma_j \in \{\tau_j, (1 - \tau_j)\}$ **do**
    for every possible terminal outcome o, with payoff $u_i(o)$ **do**
      payoff += $\alpha_k \cdot u_i(o) \cdot$ probability outcome o is attained when player i takes action a_i and other players follow $\{s_j^*\}$
      sum is incremented by same excluding $u_i(o)$ factor
  if $t \geq 1$ **then**
    for every possible hostility state $G_{h'} \neq G_h$ **do**
      p' = total probability we will transition to state $G_{h'}$ when player i takes action a_i and opposing players have type distribution $\{\tau_j\}$ and follow strategies $\{s_j^*\}$
      $\{\tau_j'\}$ = new type distributions computed using Bayes' rule assuming player i takes action a_i and the game transitions to state $G_{h'}$
      payoff += $p' \cdot$ CV$(i, t_i, \{\tau_j'\}, \{s_j^*\}, G_{h'}, t - 1)$
      sum += p'
  payoff = payoff / sum
  if payoff $>$ max-payoff **then**
    max-payoff = payoff
return max-payoff

For the case where the prior type distribution is uniform (all values equal to $\frac{1}{|T_i|}$, which is what we use in our experi-

ments), we apply Algorithm 2 as follows. For each player i and each type $t_i \in T_i$ we apply Algorithm 2, assuming that each opposing player has type t_j with probability $\tau_j = \frac{1}{|T_j|}$, using the initial game state G_0 and time horizon t. Call the result V_{t_i}. Then the optimal value for player i is $V_i = \frac{\sum_{t_i} V_{t_i}}{|T_i|}$. We repeatedly compute V_i for $t = 0, 1, 2 \ldots$ until it (hopefully) converges. We can then compare the converged values of V_i for each player to the expected payoff for the player under the computed strategy profile, $V_i^* = u_i(s^*)$. We then define $\epsilon_i = V_i - V_i^*$, and $\epsilon = \max_i \epsilon_i$. Several implementation enhancements were applied to improve the efficiency of Algorithm 2 for the IIHG. These included precomputing tables of coefficients for transition and terminal payoff probabilities and type indices, only iterating over future states with $h' > h$, and ignoring states $G_{h'}$ with extremely small transition probability from G_h (e.g., below 0.01)

New improvements for best response computation

Note that the procedure previously described involves performing $\sum_{i=1}^{n} |T_i|$ applications of Algorithm 2, once for each type $t_i \in T_i$ for each player $i \in N$. If we assume that all of the $|T_i|$ are equal, say to $|T|$, then we will call Algorithm 2 $n|T|$ times. For our specific problem we have $n = 4$ and $T_i = 2$ for all i, so we would run the algorithm 8 times. We could straightforwardly parallelize the approach by running these computations in parallel instead of sequentially. This would achieve a speedup of the runtime by a factor of $\sum_{i=1}^{n} |T_i|$ if all of the computations take equal time. However, it is possible that one of the computations takes significantly longer than the others in which case this simple parallelization would not result in a significant speed improvement. In general, given access to C cores, we would like to come up with an approach that exploits parallelism as much as possible using to minimize the total runtime.

To start let us fix $p \in N$, $t_p \in T_p$, and consider the problem of running Algorithm 2 for player p with type t_p assuming we are allocated C_p cores. This requires iterating through all nodes in a tree where the nodes correspond to belief states that are reachable from the initial state and the edges correspond to actions for player p. At each nonterminal node there are $|M_p|$ actions available. One approach would be to solve separately for each of the $|M_p|$ subtrees on its own core and select the corresponding action producing highest expected payoff. If all of the subtrees have equal size, then this would improve the runtime by a factor of $|M_p|$; however, if one of the subtrees is significantly larger than all the others then this parallelization would prove little improvement. We could extend this approach by solving separately for all subtrees that are rooted at nodes at depth 1, of which there are $|M_p|^2$, and in general solve separately for all $|M_p|^{d+1}$ subtrees rooted at depth d (as long as $C_p \geq |M_p|^{d+1}$), propagating values up the tree.

Now suppose we are at a given node v^* with children $v_1, \ldots, v_K$, and suppose that for each v_i we know exactly how many nodes are in the subtree rooted at v_i: denote this by N_i. Suppose we have C^* cores to allocate between the entire subtree rooted at v^*. Let b_{ij} be a binary variable that is 1 if core j is used for node v_i. Let Q_i denote the total number of cores that are allocated to node i. An integer program for determining the optimal allocation of cores to nodes is provided below. The first constraint ensures that the objective is to minimize the maximum amount of computation assigned to each core, where we assume that computation is divided equally between all nodes assigned to a given core. The second constraint ensures that we don't assign more cores to a node than the number of nodes in its subtree. And the third constraint ensures that Q_i is equal to the number of cores allocated to node i, as is its intended definition. Note that the first set of constraints involves division by the variables Q_i. We can define a new (continuous) variable R_i by the constraint $R_i Q_i = 1$, and therefore the value of R_i will equal $\frac{1}{Q_i}$, transforming the constraint into $u \geq \sum_{i=1}^{C^*} (N_i R_i b_{ij})$. This constraint now just involves the product of two variables, and is therefore quadratic. The overall formulation is now a mixed-integer quadratically-constrained program (QCP). Note that the first set of constraints is not convex, making the problem challenging to solve. Fortunately Gurobi has recently released an approach that is able to solve non-convex programs with quadratic objective and constraints (Gurobi Optimization 2019), which we can apply to our problem.

$$
\begin{aligned}
\min_u \quad & u \\
\text{s.t.} \quad & u \geq \textstyle\sum_{i=1}^{C^*} \frac{N_i b_{ij}}{Q_i} \text{ for all } j \\
& Q_i \leq N_i \text{ for all } i \\
& \textstyle\sum_{j=1}^{C^*} b_{ij} = Q_i \text{ for all } i
\end{aligned}
$$

We could apply this method recursively to determine the optimal allocation of cores to subtrees. Note that this approach assumes that we know in advance the values N_i denoting the number of nodes in each subtree. If it is not possible to obtain these exactly, they could be approximated by repeatedly sampling from the tree.

Due to practical challenges of implementing this approach, we also consider a simpler approach for parallelizing Algorithm 2. When we are solving for player i with type t_i for the initial state G_0 with horizon T, we can assign the subtree corresponding to each initial action a_i to a different core. This would require $|M_i|$ cores, and would result in a speedup by a factor of $|M_i|$ if all subtrees took the same amount of time.

Another improvement to Algorithm 2 can be made if we know the diameter of the tree in advance. Rather than computing V_i for $t = 0, 1, 2, \ldots$ until it converges, we can just set $t = T^* - 1$, where T^* is the diameter of the tree. Note also that the prior approach is not theoretically sound; it is possible that V_i remains the same for several values of t without the algorithm converging (for example, we may require $t = 50$ steps to reach a terminal state, and V_i may be identical for $t = 15, 16, 17$, etc.).

To summarize, we can expect to obtain a speedup of approximately $\sum_{i=1}^{n} |T_i|$ by solving for all players and types in parallel, and a further speedup by a factor of $|M_i|$ if we assign a different core to each action at the initial node for

each of these computations. Assuming that all of these computations are roughly equal, this would result in a speedup of $\sum_{i=1}^{n}(|M_i||T_i|)$, or $n|T||M|$ if all $|T_i|$ equal $|T|$ and all $|M_i|$ equal $|M|$, assuming access to $\sum_{i=1}^{n}(|M_i||T_i|)$ cores.

Experiments

We ran experiments on the strategies computed by ST-PIFP-TDV after 10 iterations for $K = 100$. We assume that $T_i = \{1, 2\}$ for all players i. For simplicity we compare the approaches for computing player 1's best response with type 1 (note that the other calculations can be performed analogously). We first present results for Algorithm 2. Recall that the limitations of this algorithm are that it is sequential, and that it makes the assumption that it ignores transitions that have probability below some threshold δ (in prior experiments $\delta = 0.01$ was used). We computed that the diameter of the tree is $T^* = 13$, so we can just solve with $t = 12$.

In Table 1, we compare the runtimes of Algorithm 2 and the parallel approach that assigns different cores for each of the initial actions. Note that we are just considering player 1 with type 1 and $t = 12$, and that the parallel approach additionally benefits by solving for all i, t_i in parallel. We are also ignoring the additional runtime of Algorithm 2 that would be due to iterating over different values of t. From the table, we can see that for $\delta = 0.00001$ we obtain a speedup by a factor of 4.4. (Note that $|M_1| = 9$.) The parallel approach is able to solve the problem exactly (using $\delta = 0$) in 26.55 hours, while the prior approach was not able to solve the problem within 45 hours. If we implemented the optimal parallelization based on the mixed-integer QCP described above we could obtain a significant further speedup.

δ	Algorithm 2 runtime	Parallel runtime
0.01	3.247	1.506
0.001	5.183	2.211
0.0001	65.403	24.031
0.00001	14952.307	3457.727
0	Did Not Finish	95589.746

Table 1: Running time in seconds for prior algorithm and new parallel algorithm for different values of δ.

Conclusion

Computing a best response is a fundamental task in game theory. One important use is to evaluate the quality of Nash equilibrium approximations. While it can often be performed in polynomial time, in imperfect-information stochastic games it is equivalent to solving a POMDP, which is PSPACE-complete. We developed two new approaches for utilizing parallelization to improve the runtime of a prior approach; the first is an optimal approach based on a mixed-integer QCP, and the second is a simpler approach that parallelizes over actions taken at the initial game state. We show that the approaches lead to a significant improvement in runtime for a 4-player imperfect-information naval planning problem. We expect these approaches to be useful for significantly improving the runtimes for a variety of sequential-graph searching algorithms, including approaches for solving POMDPs.

References

Brown, G. W. 1951. Iterative solutions of games by fictitious play. In Koopmans, T. C., ed., *Activity Analysis of Production and Allocation*. John Wiley & Sons. 374–376.

Cassandra, A. R.; Kaelbling, L. P.; and Littman, M. L. 1994. Acting optimally in partially observable stochastic domains. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Chen, X., and Deng, X. 2006. Settling the complexity of 2-player Nash equilibrium. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*.

Chen, G.; Shen, D.; Kwan, C.; Cruz, J.; Kruger, M.; and Blasch, E. 2006. Game theoretic approach to threat prediction and situation awareness. *Journal of Advances in Information Fusion* 2(1):1–14.

Conitzer, V., and Sandholm, T. 2006. Computing the optimal strategy to commit to. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*.

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

Ganzfried, S., and Sandholm, T. 2008. Computing an approximate jam/fold equilibrium for 3-player no-limit Texas hold 'em tournaments. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Ganzfried, S., and Sandholm, T. 2009. Computing equilibria in multiplayer stochastic games of imperfect information. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*.

Ganzfried, S.; Laughlin, C.; and Morefield, C. 2020. Parallel algorithm for Nash equilibrium in multiplayer stochastic games with application to naval strategic planning. In *Proceedings of the International Conference on Distributed Artificial Intelligence (DAI)*.

Ganzfried, S. 2021. Computing Nash equilibria in multiplayer DAG-structured stochastic games with persistent imperfect information. In *Proceedings of the Conference on Decision and Game Theory for Security (GameSec)*.

Gurobi Optimization, L. 2019. Gurobi optimizer reference manual.

Puterman, M. L. 2005. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Vorobeychik, Y., and Singh, S. 2012. Computing Stackelberg equilibria in discounted stochastic games. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Vorobeychik, Y.; An, B.; Tambe, M.; and Singh, S. 2014. Computing solutions in infinite-horizon discounted adversarial patrolling games. In *International Conference on Automated Planning and Scheduling (ICAPS)*.