Approximately Solving Sequential Games With Incomplete Information

Hala Mostafa and Victor Lesser

Computer Science Department
University of Massachusetts
Amherst, MA 01003-4610
{hmostafa,lesser}@cs.umass.edu

Abstract

Games of incomplete information are notorious for their difficulty which usually makes finding an exact solution intractable. The problem is even harder when the game has multiple stages and sequential decision-making is required. Some efforts have therefore addressed the issue of finding approximate equilibria for various classes of games, but these have mostly been restricted to 1-stage games. We propose an anytime search algorithm for calculating ϵ-Bayes-Nash equilibria in sequential games of incomplete information. Our algorithm first collapses the game tree by making "obvious" decisions and backing up values wherever possible. It then searches the space of strategy profiles by iteratively improving a random starting point until either the desired level of stability is reached or no further improvement is possible, in which case the point is randomly perturbed and the process repeats. We identify a continuum of measures for the stability of a strategy profile that vary in their accuracy and ease of calculation. We use one such measure to obtain results of the performance of our algorithm on randomly generated game trees and compare it to the Quantal Response Equilibria algorithm. We also present results of both algorithms on games derived from random instances of the database view maintenance problem. These experimental results demonstrate our algorithm's attractive anytime behavior which allows it to find good-enough solutions to large games within reasonable amounts of time.

1 Introduction

Games of incomplete information are notorious for their difficulty which usually makes finding an exact solution intractable. The problem is even harder when the game has multiple stages and sequential decision-making is required. We propose a general anytime algorithm for approximately solving sequential games of incomplete information by searching the space of strategy profiles for an approximate equilibrium. Our algorithm has three novel features: 1) it collapses the game tree as a pre-processing step, resulting in more tractable trees; 2) it generates local measures that guide the search by indicating which parts of a strategy profile are least stable (and therefore yield the most improvement if remedied); 3) it proposes a continuum of global measures of the stability of a profile by calculating upper bounds on players' regrets when playing this profile.

Our algorithm is different from work in the area of incomplete information games in the following respects: 1) it deals with sequential rather than the 1-shot or repeated games which have attracted the most attention so far; 2) it does not make assumptions about the interaction graph [6, 12, 13, 15] of the players (i.e. no assumptions about how or which players interact with each other); 3) unlike domain-specific algorithms (e.g. [7, 3]), ours is a general algorithm for solving general games of incomplete information. Our algorithm demonstrates attractive anytime behavior which allows it to find good-enough solutions to large games within reasonable amounts of time.

In addition to testing our algorithm on general game trees, we also test it on game trees derived from instances of the view maintenance problem. A database view is a dynamic, virtual table composed of the result set of a query executed over one or more data sources. The view maintenance problem [1, 2, 5] concerns how a view is refreshed when its underlying data sources are updated. When there are *self-interested* database managers from different institutions, the view maintenance problem can be formulated as a game; database managers decide how much information they contribute to refreshing the view based on the cost of disclosing various pieces of information and the reward of maintaining the view [10]. Because maintaining the view takes place over a number of time steps, we have a sequental game. The fact that a database manager does not know how much information others possess makes it a game of incomplete information.

This document is organized as follows. Section 2 gives a brief background on games and equilibria, presenting the problem of finding an equilibrium as a search process. Our anytime search algorithm is presented in Section 3. Section 4 shows experimental results. Related work is discussed and compared in Section 5. We conclude and discuss areas of future work in Section 6.

2 Games and Searching for Equilibria

2.1 Background: Games and Their Solutions

Non-cooperative game theory focuses on situations where self-interested agents make decisions that affect their own and each other's rewards. A solution is a *strategy profile* that prescribes, for each agent, what it should do under every possible contingency in the form of a probability distribution over actions available at that contingency. An *equilibrium* profile is one where no player stands to gain by unilaterally deviating from the strategy prescribed to it by the profile. In games of incomplete information, each player has private information, his *type*, that affects his own payoffs but is unknown to other players. The prior probability distribution over agents' types is, however, common knowledge. There are two sources of uncertainty: 1) a player does not observe the chance moves that assign other players' types and 2) the rules of the game may stipulate that certain actions by other players are not observable to it. The player may therefore be unable to distinguish among a set of nodes, an *information set*, in the game tree which have the same observable history from its prespective. Consequently, a strategy maps information sets, rather than specific nodes, to actions.

In *sequential games*, players take moves after observing moves of chance (e.g., a roll of a die) and moves of the other players. Sequential games are also referred to as *extensive form games* (EFGs). An EFG consist of multiple stages where each stage is a game. Actions taken at a stage affect the game that will be played at the next stage, thereby making it necessary to think about long-term consequences of actions. An EFG is a tuple $< I, V, E, P, H, u, p >$ where:

- I is the set of players
- The pair(V, E) is a finite directed tree with nodes V and edges E and Z is the set of terminal nodes
- $Player : V \setminus Z \rightarrow I$ determines which player moves at each decision node. $Player$ induces a partition over $V \setminus Z$ and $Player_i = \{x \in V \setminus Z | Player(x) = i\}$ is the set of nodes at which player i moves
- $H = \{H_0, ..., H_n\}$ is the set of information sets, one for each player. Each H_i is a partition of $Player_i$. The information set of a node x is denoted as $h(x)$
- $A_i(h)$ is the set of actions available at information set $h \in H_i$
- $u : Z \rightarrow \mathbb{R}$ is the utility function defined over the set of terminal nodes. For $x \in Z$, $u_i(x)$ is the payoff to player i if the game ends at node x
- p is the transition probability of chance moves

In incomplete information games, the first n levels of the tree represent chance nodes where at level i, Nature assigns player i's type with probability specified by the commonly known probability distribution over i's type space. A *strategy* σ for player i is a complete plan covering all possible contingencies for every possible type. For each information set $h \in H_i$, a *behavior strategy* is $\sigma_i(h) \in \Delta(A_i(h))$ where $\Delta(A_i(h))$ is the set of all probability distributions over actions available at information set h. A *strategy profile* is a set of strategies $\sigma = (\sigma_1, ..., \sigma_n)$, one per player. We write σ_{-i} to denote the set of strategies of all players except i.

In a perfect information game, a strategy profile σ is a Nash equilibrium if $u_i(\sigma_i, \sigma_{-i}) \geq u_i(\sigma'_i, \sigma_{-i})$ for all $i \in I$ and all σ'_i. A *Bayes-Nash* equilibrium (BNE) of a game of incomplete information Γ corresponds to the Nash equilibrium of the normal form game derived from Γ.

2.2 Searching The Space Of Strategy Profiles

2.2.1 A strategy profile as a point in space

As mentioned earlier, at each $h \in H_i$, σ specifies a probability distribution over actions available at information set h. It is therefore straightforward to think of a strategy profile as a point in multi-dimensional space. The dimensionality of the space is $\sum_{i=1}^{n} \sum_{h \in H_i} (|A_i(h)| - 1)$ where each dimension extends from 0 to 1. For each player i, for each of his information sets h, there is a dimension for each action available to i at h, except the last action which is assigned the probability left over from the other actions. Because probabilities of actions at an information set must add up to 1, not all points in the space correspond to valid strategy profiles. The search for a Bayes-Nash Equilibrium (BNE) is a search in this multi-dimensional space for a point that satisfies the equilibrium condition: given the other player's part of the profile represented by the point, no player would like to deviate from its strategy. This view was adopted in pervious work dealing with 1-stage games (e.g. [15, 11]) but large multi-stage games pose new challenges to the search algorithm.

2.2.2 Constraints on a BNE

A point in the above multi-dimensional space is a BNE if it satisfies certain constraints which guarantee that at each information set of each player, the player's strategy is rational given the expected values of actions available at that information set. In other words, if there is a single action with maximum expected value, that action is played with probability 1. If there are several such actions, the probability mass is divided among them. At a BNE, this division is such that the same rationality holds for the other player. This ensures that no player has any motivation to deviate from the prescribed strategy. Stated more formally, the following condition should hold at each information set h:

$$\sum_{a \in A(h)} \sigma_i(h,a) * E(Payoff_i(a)) = max_a(E(Payoff_i(a)))$$

where $\sigma_i(h,a)$ is the probability that strategy σ assigns to taking action a at h and $E(Payoff_i(a))$ is the expected value, to player i, of taking action a. This expected value is calculated in terms of the payoffs of the leaf nodes that i can end up in when taking action a and the probabilities of actions along the branches from the root to these leaves passing through a.

2.2.3 Approximate BNEs

It is straight forward to see that a point satisfying the above constraint at each information set is indeed a BNE. However, consider the situation where some of these constraints are violated.

For example, consider an information set h where the above equation does not hold; the right-hand side is greater than the left-hand side by 0.5. This means that, holding the other player's strategy fixed, this player gains 0.5 if he switches to the action that maximizes the right-hand side. We refer to the amount by which a constraint c is violated as δ_c, also known in the literature as *regret*.

As will be seen in Section 5, a search for an exact equilibrium corresponds to a Constraint Satisfaction Problem. The search for an approximate equilibrium where some δs are non-zero can be thought of as a Constraint Optimization Problem (COP). In both cases, the variables are the probabilities assigned to actions by strategies and the constraints are as described above. In this work, we try to find an approximate equilibrium by solving a COP.

3 Anytime Algorithm for Computing Approximate BNE

In this chapter we propose a simple algorithm that first collapses the game tree by making "obvious" decisions and backing up values wherever possible. The algorithm then tries to satisfy constraints derived from the collapsed game tree "as much as possible" by generating a random initial point and iteratively improving it until either some user-defined stability measure on the strategy profile is reached or no further improvement is possible. In the latter case, the point is randomly perturbed and the process repeats. The following sections elaborate on these steps.

3.1 Collapsing the Game Tree

Our experiments in building game trees from instances of the view maintenance problem show that the size of the *raw* game tree is quite large. Examining raw trees shows that there are some nodes at which decision making is not complicated by the incompleteness of information. These are nodes where a player would choose to reveal the same piece of information, regardless of the type of the other player. We therefore collapse the raw tree using the following simple algorithm. Initially, all nodes are roots of collapsible subtrees. The main idea is to work from the leaves of the tree upward, finding out which nodes are indeed roots of collapsible subtrees. For each such node, collapse its subtree using simple backups. The node becomes a terminal node whose payoffs reflect backed up values. Algorithm 1 shows how this is done.

```
for all $level$ such that $0 \leq level \leq 2T$ do
  collapsible[$level$] = non-terminal nodes at depth $level$
end for
for all $level$ such that $0 \leq level \leq 2T$ do
  for all $node$ in collapsible[$level$] do
    $turnPlayer$ = Player($node$)
    if $((|h(node)| == 1) \vee$
       $(|A_{turnPlayer}(h(node))|{==}1) \vee$
      (best action is the same across $h(node)$)) then
      $node$.payoff = (argmax$_{c \in Children}$c.payoff.turnPlayer).payoff
      delete all children of $node$
    end if
    remove all ancestors of $node$ from their respective
    collapsible[$level$] arrays
  end for
end for
```

Algorithm 1: Simple algorithm for collapsing trees

Figure 1 shows examples of collapsing. Action nodes are in circles enclosing the number of the acting player. Terminal nodes are shown in black circles with a pair of numbers specifying the associated payoff for each player. A dotted box encloses nodes in the same information set. The three situations where a node can be collapsed are shown; 1) incompleteness of information does not affect the player's decision (the best action is the same regardless of which particular node the player is at), 2) node in a singleton information set and 3) node with a single available action. Because we work from the leaves upward, a node eligible for collapsing always has terminal children. As will be detailed when we present our experimental results, this simple collapsing algorithm is very effective for game trees derived from the view maintenance problem.

3.2 Iteratively Improving A Point

To iteratively improve a point, we need to address the following issues:

1. Which component(s) of the point (strategy profile) should we improve? Should we focus on improving individual constraints or the profile as a whole?

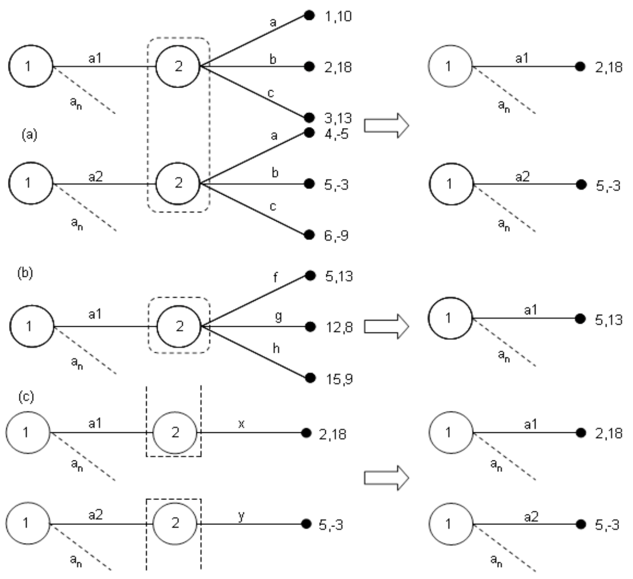

Figure 1: Collapsible Subtrees: (a) action b is the best across the information set (b) singleton information set (c) single available action

2. How should we explore the space? How do we generate neighboring points to which we can move?

3. How do we assess a point? What measure of a point indicates the algorithm is moving in the right direction in the multi-dimensional space?

3.2.1 What should we improve?

Improving individual constraints or the profile as a whole amounts to making local or global changes to a profile, respectively. A local change tries to improve a constraint associated with some information set $h \in H_i$ to reduce the regret of player i at h. A global change completely overhauls one or both players' strategies to get to a more "stable" point; one at which the players' motivations to deviate is lower. Owing to the complexity of overhauling a profile, we improve individual constraints with the hope of effecting a global improvement through local changes. Because it is not easy to determine which local changes produce the largest global improvement, we use δs as heuristics to decide which parts of a strategy profile are more important to improve. Constraints with high δs are associated with information sets with high regrets. Therefore when iteratively improving a point, our algorithm greedily attempts to improve the constraint with the highest δ first. Empirical observations indicate that this heuristic is indeed useful; improving constraints with high δs results in more stable points. We quantify the notion of stability later in the text.

3.2.2 Generating potential next points

For each variable (action probability) v involved in the constraint with the maximum δ, we calculate the required change in v to bring δ down to 0, assuming all other variables are unchanged.

We assess the impact of each potential change on the δs of other constraints by evaluating the partial derivatives of affected constraints w.r.t. v. A change that results in a point with greater than or equal stability than the current point is admitted, and the resulting point is added to the list of Potential Next Points (PNPs). Algorithm 2 shows how this is done.

```
$c$ = constraint with maximum $\delta_c$. $c = f(v_1, ..., v_k)$
**for all** $j$ in 1..k **do**
  $\Delta_j$ = required change in $v_j$ to make $\delta_c = 0$
  $\sigma'$ = point obtained by making change $\Delta_j$
  **if** stability($\sigma'$) $\geq$ stability($\sigma$) **then**
    PNP.add($\sigma'$)
  **end if**
**end for**
```

Algorithm 2: Generating potential next points

The approach described above is actually one of two ways we try to decrease a given δ_c. Instead of changing the probabilities of actions involved in a constraint as done above, we can switch the player's preference for two actions a and b by switching their probabilities. We generate points from such reversals and as with the first approach, we assess the broader impact of the change and decide whether to admit the points to PNP.

3.2.3 Assessing a point

Now that we have a set of PNPs, we need to move to the most stable PNP. Even though δs determine which part of the profile to improve first, these local measures do not provide good basis for comparing the stability of different points. The problem is that each δ specifies the additional reward a player gets if it deviates at a single information set. This does not say anything about the player's potential gains if it deviates at multiple information sets. We therefore need a global measure that specifies a player's overall motivation to deviate from (or completely overhaul) its strategy.

Following the notion of ϵ-BNE, we consider a profile stable if no player stands to make more than ϵ% more reward by deviating from (or completely overhauling) its prescribed strategy. We define a global measure called *Maximum Overall Motivation*(MOM) to deviate. MOM(σ) is the maximum, over all players, upper bound on motivation to deviate from σ assuming strategies of other players are held constant. MOM is therefore an upper bound on ϵ. The lower the MOM, the more stable σ is. Approximating an equilibrium this way makes sense because practically, a player will not want to take on the difficult task of calculating its best response strategy if it knows that it stands to increase its payoff by no more than ϵ%. We capitalize on this "inertia" and offer each player a "good enough" response to the others' strategies.

We propose a simple way of calculating MOM. To calculate the upper bound on the motivation of player i at point σ, we build a modified game $\Gamma_{revealed}$ from the original game Γ. $\Gamma_{revealed}$ is a single-player perfect information game where i plays with Nature which we construct as follows. Each node $n \in h$ where $h \in H_j$ and $j \neq i$ is changed to a chance node where the probability of Nature playing action a is $\sigma(h, a)$. In addition, the information sets in the original game tree are revealed, i.e. i is granted full access to the history of play including the moves of Nature that determined players' types, thereby removing i's uncertainty about where it is within a given information set. $\Gamma_{revealed}$, being a perfect information game, can be solved by doing simple backups. i's payoff in $\Gamma_{revealed}$ is an upper bound on the payoff of its best response strategy in Γ, since i can do no better than having perfect information. Because of the simplicity of doing backups, we can quickly evaluate MOMs for a large number of PNPs.

To summarize, we use a local measure (δ) to generate PNPs and a global measure (MOM) to assess and compare points. The global measure indicates how stable a point is, but does not give indication of how it should be improved. The local measure indicates where it may be effective to try to improve.

A range of approximations

As detailed above, we regard the problem of finding a BNE as a search in multi-dimensional space for a point/strategy profile whose regret is guaranteed to be below a certain threshold. Because the actual regret is expensive to compute, we examine the landscape given by MOM, which, at every point, is guaranteed to be no lower than the actual regret. However, examining the MOM landscape in some of the experiments we conducted showed that sometimes the upper bound provided by MOM is very loose; e.g., the MOM landscape is everywhere higher than 20%. The reason for this looseness is that the game $\Gamma_{revealed}$ is *too easy* compared to Γ. We can obtained a tighter upper bound on actual regret if we can calculate payoffs in a game that is harder than $\Gamma_{revealed}$ but still easier than Γ. In fact, there should be a whole spectrum of such games with varying levels of difficulty. At one end of the spectrum is $\Gamma_{revealed}$ where *all* information sets are disclosed. These games are very easy to solve but provide very loose upper bounds. At the other end is Γ where *no* information sets are disclosed. As we already know, these games are hard to solve.

To illustrate the kind of bound we get from a slightly harder game than $\Gamma_{revealed}$, consider the game Γ_{LI} which differs from $\Gamma_{revealed}$ only in that all information sets except the highest-level information sets for each player are revealed. Clearly, the payoff in this game is at least as high as that obtained in Γ but no higher than in $\Gamma_{revealed}$. At the same time, solving this game is not too hard; do regular backups from the bottom of the game tree upwards, and, on reaching the highest-level information sets, choose the action that maximizes reward in expectation over the turn player's beliefs about where it is within this information set. We call the maximum motivation to deviate from σ in this game *MOM-LessInformation*(MOM-LI).

To illustrate the different possible payoffs with an example, Table 1 shows the payoffs achieved by player i's different strategies when its opponent plays its part of the strategy profile $\sigma =< \sigma_i, \sigma_j >$ in different variants of an example game. *Payoff(σ)* is i's payoff from playing σ_i in the original game Γ. *PBR* is i's payoff from its best response to σ_j in Γ. It is obtained by calculating i's payoff in a transformed game where j's nodes are changed to chance nodes with action probabilities as dictated by σ_j. *PLI* is i's payoff from its best response to σ_j in the slightly harder game Γ_{LI} with less-than-perfect information. *PPI* is i's payoff from its best response to σ_j in the perfect information game $\Gamma_{revealed}$. Since PBR cannot exceed PPI or PLI, the overall motivation to deviate calculated using either of these measures is an upper bound on a player's actual regret. MOM and MOM-LI are the maximum, over all players, percentage regrets calculated using PPI and PLI, respectively. It is easy to see that the following relation holds:

$$Payoff(\sigma) \leq PBR \leq PLI \leq PPI$$

Table 1: Calculating MOM With Different Levels Of Information

Quantity	Player 1	Player 2
Payoff(σ)	8.97	7.88
Payoff of B.R.(PBR)	10.39	8.48
Payoff in $\Gamma_{revealed}$ (PPI)	12	9.09
Payoff in Γ_{LI} (PLI)	11	9
Overall Motivation (PPI-Payoff(σ))/PPI * 100%	25.25%	15.26%
MOM	25.25%	
Overall Motivation LessInfo (PLI-Payoff(σ))/PLI * 100%	18.45%	12.4%
MOM-LI	18.45%	

4 Experimental Results

4.1 The Effect of Collapsing

The first set of experiments investigates the efficacy of our collapsing algorithm for trees from random instances of the view maintenance problem (henceforth called VM trees) as well as general trees. Table 2 shows the result of collapsing VM trees. Unless indicated otherwise in brackets, we generated 10 random instances per configuration for a total of 65 instances. As can be seen, the size of the collapsed tree is roughly an order of magnitude smaller than the raw tree. This pre-processing step is therefore very useful for providing our anytime algorithm with tractable input.

To see how much general game trees collapse, we generated trees where both players have the same number of types and the same number of actions is available at each information set. We generated 10 random trees for each configuration $< T, numTypes, numActions >$ where T is the time horizon $1 \leq T \leq 4$, $2 \leq numTypes \leq 3$ and $2 \leq numActions \leq 3$ (N/A entries were too large to generate). Payoffs were generated randomly in the range [0,15]. Table 3 shows the raw tree size and average percentage reduction for these configurations.

Clearly, trees derived from the view maintenance problem are much more susceptible to collapsing than general trees. To understand why this is the case, we need to remember the source of uncertainty faced by a player in a VM tree as given in the formulation in [10]. In spite of the imperfect information about player j's type, player i has *no uncertainty* regarding the *payoffs* of actions. This results in the lowest level of the tree always collapsing, making it more likely that levels higher up in the tree collapse as well (a node is eligible for collapsing only if its children are terminals).

4.2 Performance of the Anytime Search Algorithm

We now present results of the performance of our anytime algorithm and compare it to the Quantal Response Equilibria (QRE) algorithm [14] as implemented in Gambit [9]. We ran the two algorithms (anytime search and QRE), on 2 tree types (VM and general trees) using MOM

Table 2: Collapsing VM trees

T	Raw Size	Avg % Reduction
2	716	84.3
	2253	89.5
	2253	88.8
	6847	84.7
3	3608	91.2
	15423	88.9
	36232	92.9

Table 3: Collapsing general trees with 2 (top) and 3 (bottom) types per player

	#Actions=2		#Actions=3	
T	Raw Size	Avg % Reduction	Raw Size	Avg % Reduction
1	34	23.5	58	15.5
2	130	25.5	490	21.9
3	514	27.3	4378	22.3
4	2050	24.9	N/A	N/A
1	70	11.1	124	4.4
2	286	11.7	1096	7.5
3	1150	13.5	9844	7.9
4	4606	12.6	N/A	N/A

and MOM-LI for a total of 8 sets of experiments, but for space limitations, we do not report all of them here. In all our results, we are interested in the average time, in seconds, needed to reduce regret (MOM or MOM-LI) to 5%. We bin results by tree size and show the percentage of trees in each size bin for which the algorithm could reach the desired regret within the indicated time range. Note that the reported tree size is the size of the collapsed, rather than the original, tree. We omit from our tables time or size bins that were found to be empty.

For VM trees, randomly generated costs and rewards sometimes result in a tree which collapses to an empty game. This happens if, for example, it is always lucrative to disclose all information regardless of any uncertainty. Out of the 65 VM trees reported in Table 2, 52 collapse to non-empty games. For each of these 52 trees, we ran our search algorithm 3 times starting from different random points.

For general trees, we generated 73 random trees, none of which collapsed to an empty tree. Again, on each tree we ran our search algorithm 3 times starting from different random points.

Tables 4 and 5 show results of our algorithm and QRE, respectively, on VM trees when using MOM-LI. For most of the trees in any given tree size bin, our algorithm reaches the required level of regret within 100 seconds. Compared to our anytime search algorithm, QRE has comparable performance on smaller trees but fails to finish within the allocated time on a much higher fraction of larger trees than our algorithm.

Table 4: Anytime performance on VM trees with MOM-LI

Tree Size	≤ 20 secs	21-100 secs	101-500 secs	501-1000 secs	1001-1400 secs	> 1400 secs
0-200	100%	_	_	_	_	_
200-400	96.1%	_	3.9%	_	_	_
400-1000	66.7%	20.8%	8.3%	_	_	4.2%
1000-2000	28.5%	47.6%	14.3%	4.8%	_	4.8%
2000-3600	_	44.4%	48.1%	3.7%	_	3.7%

Table 5: QRE performance on VM trees with MOM-LI

Tree Size	≤ 20 secs	21-100 secs	101-500 secs	501-1000 secs	1001-1400 secs	> 1400 secs
0-200	88.9%	_	_	_	_	11.1%
200-400	94.1%	5.9%	_	_	_	_
400-1000	50%	25%	12.5%	_	_	12.5%
1000-2000	_	_	14.3%	28.6%	_	57.1%
2000-3600	_	_	22.2%	_	22.2%	55.6%

General trees proved to be more challenging than VM trees. To illustrate the difference between using MOM and MOM-LI, Table 6 shows the result of our search algorithm using MOM. For the first 3 tree size bins, the algorithm manages to reach the desired regret level most of the time. For trees with more than 800 nodes, however, the algorithm needs an unreasonable amount of time and possibly never gets to the threshold. The situation is much better when using MOM-LI. Tables 7 and 8 show the result of our search algorithm and QRE, respectively, on general trees using MOM-LI. Our algorithm performs better than QRE on smaller smaller trees and is comparable to it on larger ones. We plan to investigate what characteristics of trees derived from the view maintenance problem make them more amenable to an algorithm like ours.

Some remarks about our results are in order. First, it should be noted that there are many possibilities for finetuning the search algorithm (e.g. random restarts and changing the magnitudes of random perturbations as the search proceeds), but we leave this for future work. Second, it is important to remember that a strategy profile provides players with a plan of action for every type with non-zero probability in the game definition. Therefore we only need to run the search algorithm when the players type spaces, or the probability distributions over them, change. In the view maintenance problem, database managers can continue using a strategy as

Table 6: Anytime performance on general trees with MOM

Tree Size	Num. of Trees	≤ 20 secs	21-100 secs	101-500 secs	> 2000 secs
0-200	105	63.8%	6.7%	_	29.5%
200-400	75	56%	22.7%	8%	13.3%
400-600	9	11.1%	77.8%	11.1%	_
800-1100	30	_	10%	10%	80%

Table 7: Anytime performance on general trees with MOM-LI

Tree Size	Num. of Trees	≤ 20 secs	21-100 secs	101-500 secs	501-1000 secs	1001-1400 secs	> 1400 secs
0-200	105	96.2%	2.9%	–	–	–	0.9%
200-400	75	85.3%	12%	2.7%	–	–	
400-600	9	44.4%	55.6%	–	–	–	–
800-1100	30	–	20%	40%	20%	–	20%

Table 8: QRE performance on general trees with MOM-LI

Tree Size	Num. of Trees	≤ 20 secs	21-100 secs	101-500 secs	501-1000 secs	1001-1400 secs	> 1400 secs
0-200	105	97.1%		–	–	–	2.9%
200-400	75	40%	56%	–	–	–	4%
400-600	9	–	100%	–	–	–	–
800-1100	30	–	–	80%	10%	–	10%

long as the number of potentially relevant tuples and the probability distributions over them are unchanged. So the time taken to calculate a strategy profile is amortized over all the view maintenance episodes for which the profile is valid.

5 Related Work

One of the few works concerned with large sequential games of incomplete information is that by Gilpin and Sandholm [4]. They automatically abstract games in such a way that any equilibrium in the smaller (abstracted) game corresponds directly to an equilibrium in the original game. However, the original game must satisfy the condition of having an ordered signal space so we cannot compare performance of our algorithm to theirs.

The idea of finding an exact (resp. approximate) equilibrium as a constraint satisfaction (resp. optimization) problem was explored in previous work, but only for 1-stage games. This line of work was started by Kearns et al for games of perfect information where the interaction graph depicting which players affect each other is a tree [6]. The basic idea was extended to games with incomplete information [12], to perfect information games with arbitrary interaction graphs [15] and to games with incomplete information and arbitrary interaction graphs [13]. These algorithms are decentralized, which allows scaling w.r.t. the number of players. However, they are all limited to 1-stage games. In the CSPs and COPs derived from these games, a variable represents a players strategy, i.e. a probability distribution over its actions. In sequential games, however, a players strategy is a probability distribution over a sequence of actions. The space of strategies is therefore enormous, even if discretized as suggested in [6].

The work of Vickrey and Koller [15] is the closest to ours. Their hill-climbing algorithm tries to minimize the sum of regrets over all players. They iterate over the steps of choosing a strategy to change, calculating the new strategy and updating affected regrets in a way that is close to ours. However, this work is again not directly comparable to ours because 1) they operate on 1-stage game and if we represent our games as 1-stage games where each strategy is actually a sequence of actions, the resulting matrix is going to be very large; 2) Vickrey and Koller primarily address a problem different from ours; scaling with the number of players by trying to exploit local interactions among them to calculate equilibria more efficiently. Another work addressing 1-stage games with complete information is that of La Mura and Pearson [11]. Their algorithm does not require that payoffs be common knowledge.

Instead of proposing an algorithm for general games, some work uses knowledge about the problem domain to solve games in this domain more efficiently (e.g. [3, 7]). However, this work is usually limited in applicability because it relies on features of specific domains.

The Quantal Response Equilibria algorithm was devised by Turcoy [14] whose work is based on that of McKelvey and Palfrey [8]. It applies a quantal choice model when the players observe payoffs with a random additive shock and choose optimally according to those noisy payoffs. In the case where the additive disturbances associated with players' strategies are drawn independently from the extreme value distribution with precision parameter λ, the form of the rule determining quantal response equilibrium choice probabilities is logistic. McKelvey and Palfrey refer to quantal response equilibria with this distribution of disturbances as logit equilibria. The set of logit equilibria can be viewed as a correspondence from λ to the set of mixed strategy profiles. At $\lambda = 0$, this correspondence contains only the centroid. As λ approaches infinity, the correspondence converges to a (possibly strict) subset of the Nash equilibria of the game.

6 Conclusion and Future Work

We propose a general anytime algorithm for approximately solving general games of incomplete information by searching the space of strategy profiles for an ϵ-Bayes-Nash equilibrium. Our algorithm has three novel features: it collapses the game tree as a pre-processing step, resulting in more tractable trees; it generates local measures that guide the search by indicating which parts of a strategy profile are least stable; it proposes a global measure of the stability of a profile as a whole by calculating upper bounds on players' regrets when playing this profile.

Experimental results demonstrate our algorithm's attractive anytime behavior which allows it to find good-enough solutions to large games within reasonable amounts of time. We conducted experiments using two types of trees; general game trees and trees derived from instances of the view maintenance problem. Collapsing general trees yields a reduction in the number of nodes in a tree by 4-27%. Collapsing is much more effective for view maintenance trees, resulting in collapsed trees roughly an order of magnitude smaller than the original trees. Experiments on finding an approximate equilibrium to the view maintenance game also produced encouraging results; our anytime search algorithm demonstrates good anytime behavior by reaching a regret level of 5% within 100 seconds for most trees and performs much better than the Quantal Response Equilibria algorithm on larger trees. General game trees proved more challenging to our algorithm. However, it still performs better than QRE on smaller trees and is comparable to it on larger ones.

The main conclusion we draw from our results is that in domains where approximate equilibria are acceptable, simple techniques prove to be effective in dealing with sequential games of incomplete information, a class of games notorious for its intractability. A simple pre-processing step significantly reduces the size of game trees, especially those derived from the view maintenance problem. A fairly simple search algorithm can navigate the huge space of strategy profiles and reach a reasonably low regret within hundreds of seconds in most cases.

We envision several potential directions for future work. Experimental results showed that our search algorithm performs better on game trees derived from the view maintenance problem than it does on general trees. We would like to gain more insight into why this is the case and what special features of view maintenance trees make them easier to solve. Another direction concerns the kind of equilibrium we search for. Currently, we search for Bayes-Nash equilibria. However, it is known in game theory literature that some of these equilibria may be unrealistic due to a phenomenon known as "incredible threats". Therefore we would like to investigate how we can search for policies that meet refined notions of equilibrium (e.g. Perfect Bayes equilibria). Yet another interesting issue we would like to investigate is whether our search algorithm can be modified to more cleverly "navigate" the policy space, something which requires some kind of description of the space's topology or a heuristic for coming up with one.

7 ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Award No. IIS-0414711. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] J. A. Blakeley, P. A. Larson, and B. W. Tompa. Efficiently updating materialized views. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, Washington, D.C., USA, 1986.

[2] C. Y. Choi and Q. Luo. Template-based runtime invalidation for database-generated web contents. In *Proceedings of Advanced Web Technologies and Applications, 6th Asia-Pacific Web Conference, APWeb 2004*, Hangzhou, China, 2004.

[3] S. Fatima, M. Wooldridge, and N. Jennings. Approximate and online multi-issue negotiation. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-agent Systems*, Hawaii, USA, 2007.

[4] A. Gilpin and T. Sandholm. Finding equilibria in large sequential games of imperfect information. In *Proceedings of the ACM Conference on Electronic Commerce*, MI, USA, 2006.

[5] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proceedings of ACM SIGMOD Conference on Management of Data*, Washington D.C., USA, 1993.

[6] M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence (UAI-01)*, CA, USA, 2001.

[7] M. Kearns and L. E. Ortiz. Maintaining views incrementally. In *Proceedings of Advances in Neural Information Processing Systems*, MA, USA, 2004.

[8] R. McKelvey and T. Palfrey. Quantal response equilibria for extensive form games. *Exper. Econ.*, 1:9–41, 1998.

[9] R. D. McKelvey, A. M. McLennan, , and T. L. Turocy. Gambit: Software tools for game theory. http://gambit.sourceforge.net/, 2007.

[10] H. Mostafa, V. Lesser, and G. Miklau. Self-interested database managers playing the view maintenance game. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multi-Agent Systems*, Estoril, Portugal, May 2008.

[11] M. Pearson and P. La Mura. Simulated annealing of game equilibria: A simple adaptive procedure leading to nash equilibrium. In *Proceedings of the International Workshop on The Logic and Strategy of Distributed Agents*, Trento, Italy, 2000.

[12] S. Singh, V. Soni, and M. Wellman. Computing approximate bayes nash equilibria in tree-games of incomplete information. In *Proceedings of the ACM Conference on Electronic Commerce*, 2004.

[13] V. Soni, S. Singh, and M. Wellman. Constraint satisfaction algorithms for graphical games. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, Hawaii, USA, May 2007.

[14] T. Turocy. A dynamic homotopy interpretation of the logistic quantal response equilibrium correspondence. *Games and Economic Behavior*, 51:243–263, 2006.

[15] D. Vickrey and D. Koller. Multi-agent algorithms for solving graphical games. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 2002.