Self-interested Database Managers Playing The View Maintenance Game

Hala Mostafa
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003-4610
hmostafa@cs.umass.edu

Victor Lesser
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003-4610
lesser@cs.umass.edu

Gerome Miklau
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003-4610
miklau@cs.umass.edu

ABSTRACT

A database view is a dynamic virtual table composed of the result set of a query, often executed over different underlying databases. The view maintenance problem concerns how a view is refreshed when the data sources are updated. We study the view maintenance problem when *self-interested* database managers from different institutions are involved, each concerned about the privacy of its database. We regard view maintenance as an incremental, sequential process where an action taken at a stage affects what happens at later stages. The contribution of this paper is twofold. First, we formulate the view maintenance problem as a sequential game of incomplete information where at every stage, each database manager decides what information to disclose, if any, without knowledge of the number or nature of updates at other managers. This allows us to adopt a satisficing approach where the final view need not reflect 100% of the databases updates. Second, we present an anytime algorithm for calculating ϵ-Bayes-Nash equilibria that allows us to solve the large games which our problem translates to. Our algorithm is not restricted to games originating from the view maintenance problem; it can be used to solve general games of incomplete information. In addition, 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.

Categories and Subject Descriptors

I.2.8 [**ARTIFICIAL INTELLIGENCE**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms, Economics

Keywords

Sequential decision-making, Games of Incomplete Information, Approximate Bayes-Nash Equilibria, Database view maintenance

1. INTRODUCTION

A database view is a dynamic, virtual table composed of the result set of a query executed over one or more data sources. A view provides a temporary, selective representation of database fields from the underlying sources. The view maintenance problem [1, 2, 4, 7, 11] concerns how views are refreshed when the data sources are updated. This problem has been extensively studied in settings where view refreshing is expensive due to factors like the communication cost of transferring large amounts of data. We study the view maintenance problem when *self-interested* database managers from different institutions are involved, each concerned about the privacy of its database. In this setting, a database manager has to decide how much it contributes to refreshing the view, and consequently how much privacy loss it suffers, based on the cost of disclosing various pieces of information and the reward received by the set of managers as a whole for maintaining the view. Because a manager's final payoff also depends on the actions of other managers, each manager needs to reason about the nature and number of updates at other databases, what they can reveal in the future and the probability of their revealing it.

The contribution of this work is two-fold. First, we formulate the view maintenance problem as a sequential game of incomplete information. Second, we propose a general anytime algorithm for approximately solving 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 preprocessing 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 global measure of the stability of a profile by calculating upper bounds on players' regrets when playing this profile.

Both of our contributions mark a departure form previous related work. Previous work on the view maintenance problem considered a setting where there is a single database manager who reveals all necessary clues in one go. Our setting is different in having multiple managers who reveal information incrementally. Also, unlike the work in [2] and [11], we reason about the tradeoff between the amount of communicated information and how up-to-date the view is in a game-theoretic way. Even when there are multiple database managers, cooperation has so far been assumed. As far as we know, we present the first work to consider the view maintenance problem with multiple self-interested database managers, a setting that is likely to increase in prevalence with the popularity of web sites drawing information from various competing sources. Some work (e.g. [10]) considers the problem of computing the result of a query de-

Cite as: Self-interested Database Managers Playing The View Maintenance Game, Hala Mostafa, Victor Lesser and Gerome Miklau, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 871-878.

fined over private datasources, a special case of the Secure Multi-party Computation (SMC) problem [19]. The difference between this kind of work and ours is that in SMC, the parties follow a protocol to carry out a computation. In our problem, each party is a *decision maker* whose contribution to different parts of the computation depends on costs and rewards; the involved parties do not follow a pre-set protocol, and thus the computation does not necessarily proceed to completion. Our problem is a decision-making problem rather than a protocol design one.

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 [8, 15, 16, 18] of the players (i.e. no assumptions about how or which players interact with each other); 3) unlike domain-specific algorithms (e.g. [9, 5]), our algorithm is not restricted to games originating from the view maintenance problem; it can be used to solve general games of incomplete information in an anytime fashion.

This paper is organized as follows. Section 2 presents the context of our view maintenance problem. Section 3 gives a brief background on games and equilibria, presenting the problem of finding an equilibrium as a search process. Section 4 shows how we formulate the view maintenance problem as a game. Our anytime search algorithm is presented in Section 5. Section 6 shows experimental results. Related work is discussed and compared in Section 7. We conclude and discuss areas of future work in Section 8.

2. VIEW MAINTENANCE WITH SELF-INTERESTED DATABASE MANAGERS

2.1 Setting

The view maintenance problem [1, 2, 4, 7, 11] concerns how database views are refreshed when base relations (tables in the source databases) are updated. We restrict our attention to views that are result sets of conjunctive queries [3] with inequalities, a class including a large number of queries used in practice. A conjunctive query is a conjunction of atomic formulae, each defined over a relation and specifies which tuples in that relation match the query. Conjunctive queries can be used to represent queries involving selections, projections and joins, a core set of relational algebra operators. In our setting, database managers (*DBMs*) maintain a view kept by a *View Holder* (VH) by disclosing information about updates made to their databases. In return, *VH* rewards the *DBM*s based on how much information they disclose. Reward is divided equally among the *DBM*s; *VH* does not care about their individual contributions.

Updates are processed in batches; view maintenance happens at intervals rather than continuously as updates are made. We assume view maintenance takes place over T time steps. Each DBM therefore has T decision points where it decides what action to take (i.e. which piece of information to disclose, if any). Different sequences of actions incur different costs for different *DBM*s and result in different final rewards. The question is which course of actions to pursue, given that each DBM wants to maximize its net profit. We investigate a DBM's tension between the self-interested behavior trying to minimize privacy and communication costs involved in disclosing updates and the cooperative behavior necessary for the *DBM*s to collect a reward.

2.2 Our Approach

Our approach has two distinguishing features. First, we regard view maintenance as an incremental sequential process where an action taken at a stage affects what happens at later stages. A DBM therefore needs to reason about the long-term as well as the immediate effects of its actions. It must also consider its uncertainty about the nature and number of updates at other *DBM*s. Second, we adopt a satisficing approach where the view need not reflect 100% of the databases updates. This creates a tradeoff between the amount of privacy given up by different *DBMs* and the quality of the view (and thus the *DBMs'* reward). Our incremental satisficing approach has the advantage of offering the *DBM*s a continuum of options rather than a binary decision of whether to fully update the view or not.

2.3 Assumptions

We make the following assumptions regarding the setting of the View Maintenance problem:

- The view query is known to all *DBM*s and information disclosed by a DBM is available to VH as well as all other *DBM*s
- A DBM incurs its own privacy/communication costs
- Rewards are divided equally among all *DBM*s regardless of their individual contributions
- The cost and reward of information are not necessarily related. The former depends on the privacy/communication cost as incurred by the revealing *DBM*. The latter only depends on the user preferences concerning the relation(s) and the type of change in question
- Tuples of the same type (e.g. inserted tuples) incur the same cost for a *DBM*. In principle, we can handle different costs, but this would result in larger trees
- The view maintenance process is invoked periodically and takes place over T time steps

3. SEARCHING FOR EQUILIBRIA

3.1 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.

3.2 Searching the space of strategy profiles

3.2.1 A strategy as a point in multi-dimensional 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 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.

3.2.2 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. 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 such that the same rationality holds for the other player. Thus no player is tempted to deviate from the prescribed strategy. Stated more formally, the following condition should hold at each information set h:

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

where $\sigma_i(h, a)$ is the probability that strategy σ assigns to taking action a at h and $E(\text{Payoff}_i(a))$ is player i's expected value of taking action a. This expected value is calculated from the payoffs of leaf nodes reachable after doing a and the probabilities of actions along the branches from the root to these leaves passing through a.

3.2.3 Approximate BNEs

What if the above constraint is violated at one or more information sets? For example, at information set h 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 can gain 0.5 by deviating from σ at h. 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 7, 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.

4. VIEW MAINTENANCE AS A GAME

4.1 Problem Abstraction

Consider 2 base relations; *Authors* and *Books* with DBMs DBM_A and DBM_B. Consider a view whose query is `"SELECT Title, Author FROM Books, Authors WHERE Pages > 600 AND Authors.City = Manhattan"` displaying the titles of all books with more than 600 pages whose authors live in Manhattan. Denoting insertion by i and deletion by d, the elements of the vector $v^j_{all} =< i^j_{all}, d^j_{all} >$ represent the number of i and d updates made to relation R_j since the last maintenance process. While v^j_{all} shows the counts of *all* the changes made, $v^j_{pr} =< i^j_{pr}, d^j_{pr} >$ shows counts for only those tuples that are judged by DBM_j to be *potentially relevant* (PR) to the view, i.e. tuples that meet the selection filter specified by the view query for R_j. Depending on whether the tuple(s) from other relation(s) that a tuple joins with (which we henceforth refer to as *complementary tuples*) meet their respective filters, the update may or may not actually be relevant to the view.

The decision problem is therefore as follows. At each of time step in the view maintenance process, each DBM decides which information to reveal; a potentially relevant changed tuple, the complement of an already relvealed tuple, or nothing. At the end of T time steps, VH rewards the DBMs based on the information revealed.

4.2 The View Maintenance Game

Our view maintenance problem can be formulated as a sequential game of incomplete information. Let n be the number of relations and assume each DBM is responsible for exactly 1 relation. Let $c \in \{i, d\}$ denote a change made to a relation, which can be insertion or deletion. Let p_c^k be the probability that a relation has k changes of type $c \in \{i, d\}$. For simplicity, we assume this probability is independent of the particular relation in question. The view maintenance game therefore has the following components [1]:

- $I = \{DBM_1, ..., DBM_n\}$
- $A_j(h)$ is the set of pieces of information that player j possesses but has not revealed on the path from the root to members of the information set h
- The type space of player j is $T_j = \{v_{pr}^j \mid 0 \leq v_{pr}^j[c] \leq v_{all}^j[c] \; \forall c \in \{i, d\}\}$; each type corresponds to a pair of possible counts of PR tuples for the 2 kinds of change. If there are m tuples as a whole affected by a given kind of change, the number of PR tuples is anywhere in [0,m]. The size of the type space is therefore $|T| = \Pi_{c \in \{i,d\}}(v_{all}^j[c] + 1)$
- The transition probability of the chance move assigning player j's type is $p \in \Delta(T_j)$ where $\Delta(T_j)$ is the set of all probability distributions over T_j. Assuming the numbers of i and d changes are independent, the probability of a type is $p(v_{pr}^j) = \Pi_{c \in \{i,d\}} p_c^{v_{pr}^j[c]}$
- The payoff $u(z)$ at a terminal node z is determined by the sequence of actions taken on the path from the root to z. We need to specify, for each action, the cost to the player disclosing the information and the common reward that all players get when this information is revealed
- T specifies the number of stages in the game

Type Probabilities and Payoff Function

We assume that initially, each DBM_j discloses its v_{all}^j. Alternatively, this information can be obtained from statistics about how many changes of each type are made to the database, on average. Specifying a probability distribution over the types of a player DBM_j (i.e. values of v_{pr}^j) can be done by estimating the selectivity of a given query, i.e. the number of tuples that match it [12].

Disclosed information has an associated cost incurred by the player who reveals it and a reward that is given to all players. We base reward on 3 factors: 1) the type of change (i or d); 2) the base relation affected by the change and 3) whether the information represents a main tuple or the complement of an already disclosed tuple. The rationale is that user preferences can be such that one type of change is more important than the other and some relations need to be more up-to-date than others. The third factor allows the VH to express different preferences for knowing different kinds of information. As in the case of rewards, disclosing different information incurs different amounts of privacy, communication and other kinds of costs. The incurred cost can also depend on what has been revealed so far (e.g. privacy costs can be sub- or super-additive).

[1]We assume that moves are sequential rather than simultaneous; a player taking an action can observe all earlier actions.

5. ANYTIME ALGORITHM FOR COMPUTING APPROXIMATE BNE

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 (a specific strategy profile) and iteratively improving it either reaches some user-defined stability measure or no further improvement is possible. In the latter case, the point is randomly perturbed and the process repeats. The following subsections elaborate on these steps.

5.1 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. We work from the leaves of the tree upward, determining which nodes are indeed roots of collapsible subtrees. For each such node, we 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 ≤ level ≤ 2T do
  collapsible[level] = non-terminal nodes at depth level
end for
for all level such that 0 ≤ level ≤ 2T do
  for all node in collapsible[level] do
    turnPlayer = Player(node)
    if ((|h(node)| == 1) ∨
      (|A_turnPlayer(h(node))|==1) ∨
      (best action is the same across h(node))) then
      node.payoff =
      (argmax_{c∈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 in Section 6, this simple collapsing algorithm is very effective for game trees derived from the view maintenance problem.

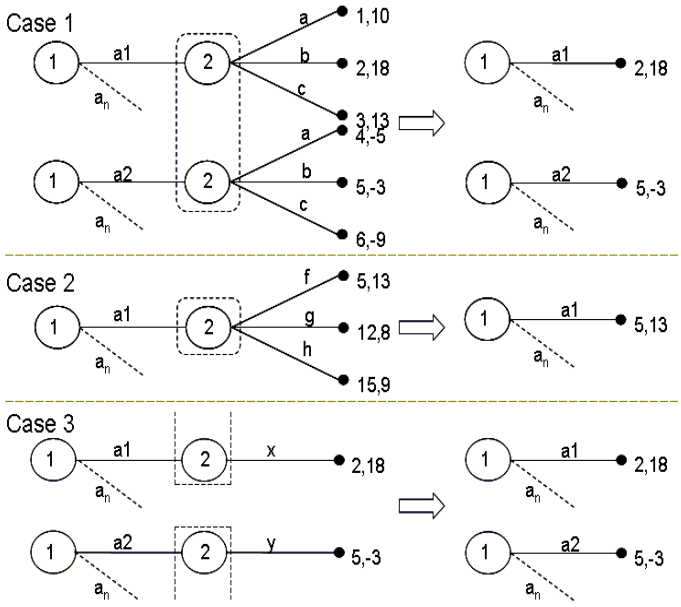

Figure 1: Collapsible subtrees: (1) action b is the best across the information set (2) a singleton information set (3) a single available action

5.2 Iteratively improving a point

To iteratively improve a point (strategy profile), the following 3 issues need to be addressed:

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

5.2.1 What should we improve?

As mentioned in Section 3.2.2, an equilibrium point/profile must satisfy certain constraints. 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 the local regret, δ, as a heuristic 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 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.

5.2.2 Generating potential next points

For each variable (action probability) v involved in the constraint c with the maximum δ_c, we calculate the required change in v to bring δ_c 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).

The approach described above is one of two ways of decreasing a given δ_c. Instead of changing the probabilities of actions involved in c 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.

5.2.3 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 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. MOM indicates how stable a point is, but does not give indication of how it should be improved. The δs indicate

Table 1: Calculating MOM With Different Amounts 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%	

where it may be effective to try to improve.

A range of approximations

Examining the MOM landscape showed that sometimes the upper bound provided by MOM is very loose; e.g., MOM is everywhere higher than 20%. The reason is that $\Gamma_{revealed}$ is *too easy* compared to Γ. We can get a tighter upper bound if we calculate payoffs in a game that is harder than $\Gamma_{revealed}$ but still easier than Γ. In fact, there is a whole spectrum of such games with varying levels of difficulty. One extreme is $\Gamma_{revealed}$ where *all* information sets are disclosed. These games are very easy to solve but provide very loose upper bounds. The other extreme is Γ where *no* information sets are revealed.

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 $\Gamma_{revealed}$. Solving this game is still easy; 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 σ 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 this relationship holds: $Payoff(\sigma) \leq PBR \leq PLI \leq PPI$, 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. Note that in the case of Table 1, we get a much tighter bound on regret when using PLI rather than PPI (18.45% vs 25.25%) at the cost of a slightly more involved computation.

Table 2: Collapsing VM trees

T	v_{all}	Raw Size	Avg % Reduction
2	$<1,1>$	716	84.3
	$<1,2>$	2253	89.5
	$<2,1>$	2253	88.8
	$<2,2>$	6847	84.7
3	$<1,1>$	3608	91.2
	$<2,1>$	15423	88.9
	$<3,1>$(5)	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

6. EXPERIMENTAL RESULTS

6.1 The Effect of Collapsing

The first set of experiments we conducted 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. Both players have the same type space (v_{all}). 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 $1 \leq T \leq 4$, $2 \leq numTypes \leq 3$ 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. To understand why this is the case, we need to remember the source of uncertainty faced by a player in a VM tree. With imperfecet information about player j's type, player i is uncertain about the number and nature of tuples yet undisclosed by j. However, there is *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).

Table 4: Performance of our algorithm(top) vs. QRE(bottom) on VM trees

Tree Size	≤ 20 sec	21-100 sec	101-500 sec	501-1000 sec	> 1000 sec
0-200	100 88.9	– –	– –	– –	– 11.1
200-400	96.1 94.1	– 5.9	3.9 –	– –	– –
400-1000	66.7 50	20.8 25	8.3 12.5	– –	4.2 12.5
1000-2000	28.5 –	47.6 –	14.3 14.3	4.8 28.6	4.8 57.1
2000-3600	– –	44.4 –	48.1 22.2	3.7 –	3.7 77.8

Table 5: Performance of our algorithm(top) vs. QRE(bottom) on general trees

Tree Size	≤ 20 sec	21-100 sec	101-500 sec	501-1000 sec	> 1400 sec
0-200	96.2 97.1	2.9 –	– –	– –	0.9 2.9
200-400	85.3 40	12 56	2.7 –	– –	– 4
400-600	44.4 –	55.6 100	– –	– –	– –
800-1100	– –	20 –	40 80	20 10	20 10

6.2 Performance of the Search Algorithm

We compared our anytime algorithm to the Quantal Response Equilibria (QRE) algorithm [17] as implemented in Gambit [13]. We ran the two algorithms (anytime search and QRE), on 2 tree types (VM and general trees) using MOM and MOM-LI for a total of 8 sets of experiments. 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 lack of space, and because results using MOM-LI are always better than using MOM, we only present the former.

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 (35 have less than 200 nodes, 25 in the range 201-400, 3 in 401-600 and 10 in 800-1100). Again, on each tree we ran our search algorithm 3 times starting from different random points.

Table 4 compares the performance of our algorithm and QRE on VM trees when MOM-LI is used. For most of the trees in any given tree size bin, our algorithm reaches the required level of regret within 100 seconds. Our anytime search algorithm performs better than QRE on smaller trees and much better than it on larger trees. QRE fails to finish within the allocated time on a much higher fraction of larger trees than our algorithm.

General trees proved to be more challenging than VM trees. Table 5 compares the result of our search algorithm and QRE on general trees using MOM-LI. Our algorithm performs better than QRE on 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.

In trying to understand why our algorithm performs poorly on certain trees, we found out that we get to points with low MOM-LI fairly rapidly (we reach a MOM-LI of 6.7% on all trees in the last row of Table 5 within 860 seconds). The main difficulty is in getting out of local minima. This suggests that the MOM-LI landscape may have a high elevation everwhere, in which case there is nothing the search procedure can do, and the only way to guarantee regret below the threshold is to further refine the stability measure.

Some remarks about our results are in order. First, there are many possibilities for fine-tuning the search algorithm (e.g. 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 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.

7. RELATED WORK

One of the few works concerned with large sequential games of incomplete information is that by Gilpin and Sandholm [6]. 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.

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, by Kearns et al for games of perfect information where the interaction graph depicting which players affect each other is a tree [8]. The idea was extended to games with incomplete information [15], perfect information games with arbitrary interaction graphs [18] and games with incomplete information and arbitrary interaction graphs [16]. 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 player's strategy, i.e. a probability distribution over its actions. In sequential games, however, a player's strategy is a probability distribution over a sequence of actions. The space of strategies is therefore enormous, even if discretized as suggested in [8].

The work of Vickery and Koller [18] 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. Another work addressing 1-stage games with complete information is that of La Mura and Pearson [14]. Instead of proposing an algorithm for general games, some work uses knowledge about the problem domain for more efficiency (e.g. [5, 9]). However, this work is usually limited in applicability.

8. CONLUSION AND FUTURE WORK

We study the view maintenance problem when *self-interested* database managers are involved, each concerned about the privacy of its database. We regard view maintenance as an incremental, sequential process and adopt a satisficing approach where the final view need not reflect 100% of the databases updates. We formulate the problem as a sequential game of incomplete information and present an anytime algorithm for calculating ϵ-Bayes-Nash equilibria. 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.

Our main conclusion is that simple techniques proved 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 reduced the size of games, especially those derived from the view maintenance problem. A fairly simple search algorithm could navigate the huge space of strategy profiles and reach a reasonably low regret within hundreds of seconds in most cases.

In future work, we will try to find out why general trees are more challenging than view maintenance trees and whether the latter have special characteristics that we can leverage.

9. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation 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 National Science Foundation.

10. 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] K. Candan, D. Agrawal, O. P. W. Li, and W. Hsiung. View invalidation for dynamic content caching in multitiered architectures. In *Proceedings of the 28th Very Large Data Bases Conference*, Hongkong, China, August 2002.

[3] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, Colorado, USA, 1977.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] G. Liang and S. S. Chawathe. Privacy-preserving inter-database operations. In *Proceedings of the Second Symposium on Intelligence and Security Informatics*, Arizona, USA, 2004.

[11] A. Manjhi, P. B. Gibbons, A. Ailamaki, C. Garrod, B. M. Maggs, T. C. Mowry, C. Olston, A. Tomasic, and H. Yu. Invalidation clues for database scalability services. In *Proceedings of the 23rd International Conference on Data Engineering*, Istanbul, Turkey, April 2007.

[12] Y. Matias, J. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, WA, USA, 1998.

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

[14] 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.

[15] 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.

[16] 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.

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

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

[19] A. Yao. How to generate and exchange secrets. In *Proceedings of the Twenty-Seventh Symposium on Foundations of Computer Science*, 1986.