# View Maintenance With Self-Interested Decision Makers[*]

Hala Mostafa, Victor Lesser, and Gerome Miklau

Computer Science Department
University of Massachusetts
UMass Computer Science Technical Report 07-55

November 2, 2007

## Contents

[*]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.

**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 $\epsilon$-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.

## 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 [3, 5, 7, 13, 23] concerns how a view is refreshed when its underlying 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 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 global measure of the stability of a profile as a whole 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 [5] and [23], 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, previous work assumed their cooperation. 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] and [21]) considers the problem of computing the result of a query defined over private datasources, a special case of the Secure Multi-party Computation (SMC) problem [35]. 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* that can accept or refuse to contribute to different parts of the computation based 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 [17, 31, 32, 34] of the players (i.e. no assumptions about how or which players interact with each other); 3) unlike domain-specific algorithms (e.g. [18, 9]), our algorithm is not restricted to games originating from the view maintenance problem; it can be used to solve 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.

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

# 2 View Maintenance with Self-Interested Database Managers

The view maintenance problem [5, 20, 23, 7] concerns how database views are refreshed when *base relations* -tables from which data populating the views is extracted- in the underlying databases are updated. This problem has been extensively studied in settings where view refreshing is expensive (e.g. because the amount of data to be communicated from databases to the view owner is large). In this section we detail the setting in which we consider the view maintenance problem and discuss the merits of the approach we take to solve it. We end the section with the assumptions we make in solving the view maintenance problem.

## 2.1 Setting

In our setting, the database managers (*DBMs*) disclose some information about their database updates in order to provide the application server responsible for the dynamic web page (henceforth referred to as the view holder or *VH*) with a more up-to-date view. In return, *VH* gives the *coalition* of DBMs a reward that depends on how much information about their respective updates they disclosed and how much is still hidden. The reward is divided equally among the *DBMs*; *VH* does not care about the individual contributions of the *DBMs*. Typically, no individual *DBM* can determine what the refreshed view should be. Different courses of actions incur different privacy costs for different coalition members. The question is which course of actions to pursue, given that each *DBM* wants to maximize its net payoff. We investigate the tension between the self-interested behavior trying to minimize privacy loss and the cooperative behavior necessary for the coalition to collect a reward.

In our setting, the updates made to the base relations are processed in batches; the process of refreshing the view happens at intervals rather than continuously as updates are made. These intervals can be fixed in length or can depend on the number of updates made (run the refresh process after $x$ updates are made). At the end of an interval, the updates made since the last refresh make up the input to the refresh process. We assume that the $VH$ gives the $DBMs$ $T$ time steps to (partially) update the view. There are therefore $T$ decision points for each $DBM$.

In this work, we are mainly concerned with conjunctive queries. Conjunctive queries were first introduced by Chandra et. al [6] and include a large number of queries used in practice. A conjunctive query takes the form $(x_1, x_2, ..., x_k)\exists x_{k+1}, ..., x_m.A_1 \wedge A_2 \wedge ...A_p$ where each $A_i$ is an atomic formula $R_j(y_1, ...y_l)$ where each $y$ is either a variable or a constant and each $R_j$ is a relation. The atomic formulas specify which tuples in a base relation match the query. An example conjunctive query is "List all departments that sell pens and pencils" given a relation Sales(Department,Item). This query would be written as $(x)Sales(x, pens) \wedge Sales(x, pencils)$.

Conjunctive queries can be used to represent queries involving selections, projections and joins, known as select-project-join queries. For example, the query "List the names of authors of books published by PHI" over relations $Books(bookID, title, publisher, authorID)$ and $Authors(authorID, name, city)$ can be written as $(x)\exists x_1, x_2, x_3, x_4.Books(x_1, x_2, PHI, x_3) \wedge Authors(x_3, x, x_4)$. Note how the join was achieved by using the same variable as the foreign key in *Books* and the primary key in *Authors*. Primary keys enforce entity integrity by uniquely identifying entity instances. Foreign keys enforce referential integrity by completing an

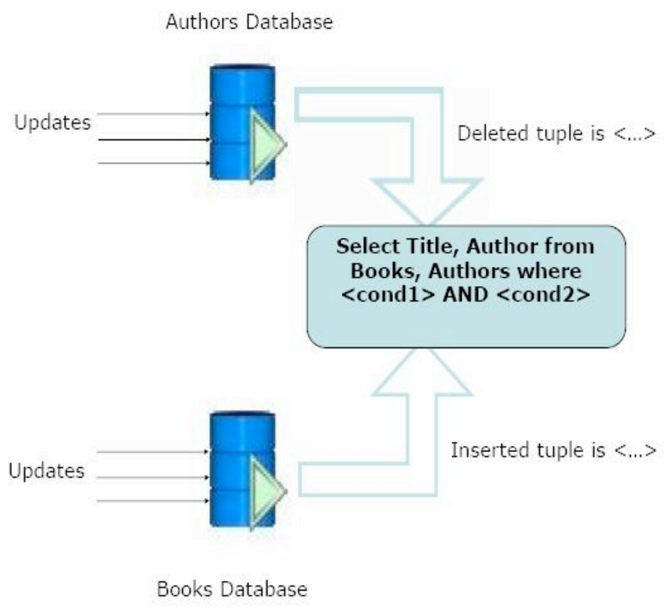

Figure 1: The View Maintenance Problem

association between two entities. The original definition of conjunctive queries does not allow for comparisons between data values, but the class of conjunctive queries was extended to include built-in predicates which are arithmetic comparisons.

## 2.2 Our Approach

Our proposed approach has two distinguishing features. First, we regard refreshing the view as an *incremental* process where at each step, each *DBM* decides what to disclose given everything the *DBMs* have disclosed so far, the cost of available actions and the expected reward from each. A *DBM* therefore needs to reason about the long-term as well as the immediate effects of its actions. An action with a low privacy cost (short term cost) could lead to a situation where actions with high cost are required (long term cost). This reasoning must deal with the uncertainty associated with not knowing the private information of the other DBMs. Second, we adopt a *satisficing* approach where the final 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 final view (and thus the *DBMs*' final reward).

Our incremental satisficing approach has the advantage of offering the DBMs a continuum of options, rather than a binary decision of whether to fully refresh the view or not. This continuum allows the DBMs to reason about the privacy-freshness tradeoff at a much finer granularity. If the privacy cost of fully refreshing the view is $C$ and the reward given by the VH at the end is $R$, then if $C > R$, an all-or-nothing approach would result in the DBMs deciding not to reveal anything. On the other hand, if it is the case that there is a partial refresh that costs $C - \epsilon_1$ and results in reward $R - \epsilon_2$ and $C - \epsilon_1 < R - \epsilon_2$, the DBMs will go ahead and provide information needed for this partial refresh.

We concern ourselves with a continuum of information that can be disclosed about the updates to the underlying databases . Along this continuum, a DBM can display any fraction of changed tuples. However, there are some satisficing behaviors that cannot be represented using our model. For example, our reward function (detailed later) cannot model a setting where the user is satisfied knowing there were more insertions than deletions, and does not care about more information regarding the particular insertions and deletions.

There are at least 2 reasons that motivate a satisficing approach in the view maintenance problem in particular, and situations where databases reveal information about themselves in general. For the view maintenance problem, the all-or-nothing approach may not be necessary if the user has different preferences for the different kinds of database updates. For example, a user can tolerate being presented with out-of-date records (i.e. not all deletions are reflected in the view), but cannot tolerate missing any tuples that have been inserted. In this case, the $VH$ can get the $DBM$s to disclose as many insertions as possible by increasing the penalty and reward associated with missing insertions. The $DBM$s will find that the way to maximize their payoffs is to disclose more insertions than other types of change.

The second example of motivating situation is not restricted to the view maintenance problem and involves any type of information disclosure. Consider an information seeking agent who needs to make a decision whether to invest in obtaining a large body of information which may or may not turn out to be useful to him. With an all-or-nothing approach, the agent is forced to make the decision up front, which involves high risk. With the ability to obtain crude information initially, followed by finer levels of detail, this agent can obtain as much information as necessary to provide him with an adequate level of confidence in his decision before potentially investing in obtaining the whole body of data. One example can be an investigator who cannot have access to a certain sensitive database without going through the process of obtaining a subpoena. If he can get partial information about this database that helps him decide whether it actually contains relevant information, he will make a more informed decision on whether or not to go with the subpoena process. Another example is a marketing firm that needs demographic data about a certain neighborhood to decide whether to launch a campaign there. Partial information about the income levels in the neighborhood can help the firm reach a better decision. Generally, an incremental satisficing approach is suitable where a decision needs to be made whether to seek more information, and the quality of this decision increases with increased knowledge about the nature of information sought.

### 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
- No individual $DBM$ can tell what the refreshed view should be
- Each $DBM$ incurs its own privacy and communication costs
- Rewards are divided equally among all $DBM$s, regardless of their contribution
- The cost and reward of a piece 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, as detailed below
- Revealing tuples of the same type (e.g. all inserted tuples) incurs the same cost for a $DBM$. In principle, our approach can handle different costs for different tuples. However, in this case we have to distinguish between the actions of revealing different tuples, which would result in a much higher branching factor of the game tree derived from an instance of the View Maintenance problem.
- The view refresh decision process is invoked periodically. Every invocation is a decision process that considers all changes that happened since the last invocation, in addition to left-over changes from the last decision process.

## 3 Games and Searching for Equilibria

In this section we give a brief background on games, their types and their solutions, with particular emphasis on sequential games of incomplete informaiton. We then describe how the process of finding an equilibrium is framed as a search process and what it means to search for an approximate equilibrium.

### 3.1 Background: Games and Their Solutions

The field of game theory focuses on situations where self-interested players (also referred to as agents) make decisions that affect each other and affect a common environment. Each agent tries to respond to the decision-making strategies of the others in a way that maximizes his own reward. These situations are typically called games. Games can be divided along several axes. Perfect recall (vs. imperfect recall) games involve players who never forget actions, whether theirs or others', once they observe them. In games of incomplete information (vs. perfect information), a player does not know what moves have already been played by other players, resulting in uncertainty about the current state of the world and multiple game situations being indistinguishable to that player. Games can also be classified by the number of stages (decision-making points) they contain; 1-stage games involve only one stage of decision making while 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.

In game theory, a solution concept is any rule for specifying predictions as to how players are expected to behave in any given game. A solution concept may specify more than one prediction for a given game, where the selection among these predictions in a specific real situation may depend on the environment . 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. The solution is usually an *equilibrium* set of strategies (according to some notion of equilibrium), one for each player, where no player stands to gain by unilaterally deviating from its strategy. The general opinion is that reaching an equilibrium is a desirable state of affairs since it provides stability to the set of players. Another advantage of playing an equilibrium strategy is that a player can do no worse, even if his strategy is revealed to the opponents. Some researchers, however, find the quest for equilibrium unjustified, as in [30]. One objection concerns the fact that there may exist multiple equilibria and simply finding one of them is not enough; we need to make sure that all players play the same equilibrium.

Formally, 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
- $Player : V \setminus Z \to I$ 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 $h \in H_i$
- $u : Z \to \mathbb{R}$ 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

## 3.2 Games of Incomplete Information

Games of incomplete information are used to model situations where each player has private information, his *type*, that affects his own payoffs but is unknown to the other players. However, the prior probability distribution over agents' types is common knowledge. Such game of *incomplete* information, where an agent is missing some information about one or more aspects of the other agents, is transformed to a game of *imperfect* information where the the agent knows some probability distribution over the missing information, but does not have perfect knowledge of what it is exactly [15]. This transformation is effected by adding random moves of Nature assigning a type for each player according to the prior distribution. A player cannot observe moves of Nature that assign other players types. Additionally, the rules of the game may stipulate that certain actions by other players are not observable by this player. As a result, a player may not be able to distinguish among a set of nodes in the game tree if all these nodes have the same observable history from this players prespective. An *information set* is a set of nodes (*members* of the information set) indistinguishable to a player. Consequently, a player makes its decision as a function of the information set, rather than the particular node, it is at.

In incomplete information games, the first $n$ levels of the game 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* $\sigma$ 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 $\sigma_{-i} = (\sigma_1, ..., \sigma_{i-1}, \sigma_{i+1}, ..., \sigma_n)$ to denote the set of strategies of all players except $i$.

**Notions of Equilibrium**

As mentioned earlier, the goal in competitive settings is typically to find a strategy profile from which no player has motivation to deviate. In games of perfect information, the Nash equilibrium is the used solution concept. A strategy profile $\sigma$ for a game $\Gamma =< I, V, E, P, H, A, u, p >$ 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 $\sigma_i'$. Some additional requirements led to the appearance of equilibrium refinements; definitins of equilibria that stipulate more conditions than the basic no-deviation condition. For example, these refinements may aim to maximize some measure of social utility, guarantee that a player acts rationally even at nodes in the game tree that would not be reached if the equilibrium profile is played, or guard against players reacting to incredible threats.

A *Bayesian Nash* equilibrium (BNE) of a game with incomplete information $\Gamma$ corresponds to the Nash equilibrium of the normal form game derived from $\Gamma$. BNE is defined as a strategy profile and beliefs specified for each player about the types of the other players. The expected payoff for each player is maximized given their beliefs about the other players' types and given the strategies played by the other players. When using this notion of equilibria, the players may reach an unrealistic equilibrium that does not make sense. The reason is a phenomenon known as *incredible threats* where a player $i$ tries to avoid a threat made by another player $j$ but the threat is implausible in that $j$ would not carry out the threat if it is playing rationally.

To remedy the incredible threats problem of NEs and BNEs, we need to ensure that players make a rational decision even at nodes off the equilibrium path. In other words, players should play rationally in every *subgmae*; a part of the game tree that does not cut across any information set. In games of complete information, every node is the root of a subgame and this equilibrium refinement is called *subgame perfect equilibrium*. In games of incomplete information, however, a game generally has only one subgame, which is the game itself. *Perfect Bayesian equilibrium* is an equilibrium refinement that specifies a belief-strategy pair that satisfy the following condition: the beliefs are consistent with the strategy and the strategy is rational given the beliefs. A player's belief is, for every one of his information sets, a probability distribution over members of the information set reflecting the player's beliefs about being at each member of the set.

## 3.3 Searching The Space Of Strategy Profiles

### 3.3.1 A strategy profile as a point in space

As mentioned earlier, at each $h \in H_i$, $\sigma$ 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.

### 3.3.2 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, either the player takes the action with the highest expected payoff with probability 1, or the player is indifferent between actions, in which case the probability distribution over these actions does not matter. This ensures that no player has any motivation to deviate from the prescribed strategy.

Because our game is a sequential one, we had to devise a set of constraints different from those used for 1-stage games (e.g. [34]). The set of constraints on a strategy profile $\sigma$ is as follows. For each information set

$h$, for every pair of actions $a$ and $b$ available at $h$, there is a constraint defined over the difference between their expected values $E(a) - E(b)$. For any action $a$, $E(a)$ is expressed in terms of probabilities of actions along all paths in the game tree that include $a$, as well as probabilities of chance moves along these paths. Based on the probabilities assigned by $\sigma$ to playing $a$ and $b$ at $h$, the type of the constraint is determined to be *GTE*, *LTE*, *EQ* or *DC* (don't care) as follows:

```
if $(0 < \sigma(h,a) < 1) \wedge (0 < \sigma(h,b) < 1)$ then type = EQ
else if $\sigma(h,a) == 1$ then type = GTE
else if $\sigma(h,b) == 1$ then type = LTE
else type = DC
```

Basically the above constraints make sure that if the player is indifferent between 2 actions, these actions should indeed have equal expected payoffs. If, on the other hand, the player chooses a certain action all the time, its expected payoff should be at least as much as that of the non-preferred action. When both actions have 0 probability, we do not care how their expected payoffs compare.

### 3.3.3 Approximate BNEs

It is straight forward to see that a point satisfying the above constraints at each information set is indeed a BNE. However, consider the situation where some of these constraints are violated. For example, what if $E(a) - E(b) = 0.5$ for a constraint of type *EQ*? In this case, the player has a motivation of magnitude 0.5 to deviate from the prescribed strategy at this information set, assuming the rest of the strategy profile (other player's strategy, as well as this player's strategy at other information sets) is unchanged. For a constraint $c$, we denote how much it is violated by $\delta_c$. For example, $\delta_c = 0.5$ if $E(a) - E(b) = 0.5$ and type=$EQ$, but $\delta_c = 0$ if type=$GTE$. $\delta$ is known in literature as *regret*.

As will be seen in the chapter on Related Work, a search for an exact equilibrium corresponds to a Constraint Satisfaction Problem. The search for an approximate equilibrium where some $\delta$s are non-zero can be thought of as a Constraint Optimization Problem (COP). In both cases, the variables are the probabilities of actions 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

In this section we detail how the view maintenance problem is formulated as a sequential game of incomplete information. We start by presenting the abstraction we will be using, then give a formal definition of the view maintenance game. We conclude the section with discussions of two elements of the game; the reward function and the type probabilities.

## 4.1 Problem Abstraction

Consider 2 base relations; *Authors* and *Books* with $DBM$s $DBM_A$ and $DBM_B$. Consider a view whose query is `"SELECT Title, Author FROM Books, Authors WHERE Books.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 $R_j$. For the conjunctive view queries we consider, a tuple $t$ in relation $R$ is potentially relevant to the view if it meets the selection filter specified by query's atomic formula for $R$. In our example, a tuple in the books relation is potentially relevant if the book has more than 600 pages. 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.

We believe elements of the vector $v^j_{pr}$ represent strategic information that $DBM_j$ would not like to reveal. The importance of this information is if $DBM_j$ has many PR tuples, it may want to withhold this fact and wait for some $DBM_i$ to disclose tuples rather than go ahead and disclose tuples itself. In this case, $DBM_i$, not knowing exactly how many PR tuples $DBM_j$ has, may worry that not enough reward is being accumulated and thus choose to disclose its own tuples. This situation is clearly to $DBM_j$'s advantage.

The decision problem is therefore as follows. At each of the $T$ time steps of the view maintenance process, each $DBM$ decides which piece of information to reveal; a potentially relevant changed tuple, the complementary tuple of an already relvealed tuple, or nothing. At the end of $T$ time steps, the $VH$ gives the $DBM$s a reward that depends 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 $|T| = \Pi_{c\in\{i,d\}}(v_{all}^j[c] + 1)$ [2]
- 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
- $T$ specifies the number of stages in the game

### 4.2.1 Types and Type Probabilities

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. We now discuss some techniques that can be used to generate the probability distribution over the types of $DBM_j$ (i.e. values of $v_{pr}^j$).

The probability of a given $v_{pr}^j$ can be estimated from historical statistics. A simple technique is keeping a running average of the number of changes made between each two consecutive refresh processes. Estimating these counts is actually of broader importance to the databases community. There has been work on estimating the selectivity of a given query, i.e. the number of tuples that match it [24, 22]. One area where such an estimate is useful is in query optimization where a database management system tries to evaluate the costs of different query execution plans. In order to do this, the optimizer needs to estimate the number of tuples that make up the inputs to the various stages of an execution plan.

Selectivity estimation can done based on very simplified and unrealistic assumptions about the underlying data distribution. Some commercial database management systems make the *attribute value independence* assumption under which a system considers attributes probabilities individually and the joint probabilities are derived by multiplying the individual probabilities. Real-life data rarely satisfy this assumption. More involved approaches use sampling-based estimations [14]. Sampling-based techniques are based on the fact that a large data set can be well represented by a small random sample of the data elements. The selectivity estimation problem gets more complicated when a query involves multiple attributes in a relation. In this

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

[2]To see how $v_{pr}^i$ affects the reward function, we follow the argument in [28] whereby information affecting the set of actions available to a player can be thought of as affecting the player's reward function. We can think of all the actions being available all the time, with the resulting payoffs depending on the private information.

case, the selectivity depends on these attributes' joint data distribution, i.e. the frequencies of all combination of attribute values [24].

### 4.2.2 Reward Function

The particular reward function used is key in deciding whether the $DBMs$ will actually end up doing what we desire them to do; refresh the view as much as possible within their individual requirements to suffer the least privacy loss possible. We considered the following different reward functions and their suitability to our setting.

- Reward based on the fraction of changes disclosed
- Reward based on the relevance of the disclosed changes to the view. This has the advantage of ignoring extraneous information and encouraging $DBMs$ to disclose those pieces of information that most contribute to the utility of the view user
- Each piece of disclosed information has an associated reward, regardless of its relevance

It is clear that the first option is not viable because the amount of change itself is unknown and cannot be determined unless all changes are actually disclosed. For the game theory formulation, the second option is not suitable either for the following reasons. First, the size of the game would be huge if reward were based on relevance. Suppose the $DBM$ of the *Books* relation $M_B$ discloses a books tuple $t_B$ affected by a certain change. Now the *Authors* $DBM$ $DBM_A$ can choose to disclose the complementary tuple of $t_B$; *ct*. To reason about this action at planning time, $DBM_A$ needs to attach a reward to it. If this reward is to depend on whether *ct* is relevant, this relevance information must be considered at planning time, i.e. $DBM_A$ has to plan for both contingencies, relevant and irrelevant. In an incomplete information game, this information would be part of a player's type. For $n$ complementary tuples, there would be $2^n$ different types necessary to represent possession of this information alone. The size of the game tree would increase significantly and it would be impractical to plan for every possible relevance contingency.

The second reason for the inadequacy of a relevance-based reward function can be explained at a higher level. In the preliminary experiments we have conducted so far, sometimes $DBM_A$ reveals a tuple that incurs short term loss in the hope that $DBM_B$ will reveal its complementary tuple and both agents get rewarded (regardless of relevance). If the reward were based on relevance, $DBM_A$ would not be so sure that a future gain will overcome its immediate loss. It does not know whether the complementary tuple will turn out to be relevant, so it will be reluctant to disclose its own tuple.

We are therefore left with the third reward function which includes 2 components, one for the privacy cost of an action and another for the reward the $VH$ associates with it. First, we discuss the reward component. We base the reward for a piece of information 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 complementary 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 pieces of 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).

# 5 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.

## 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. 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 2 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.

## 5.2 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?

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?

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

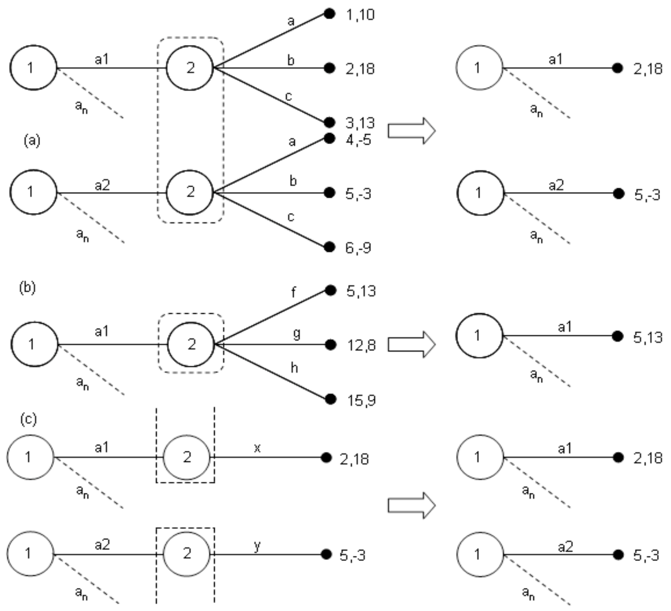

Figure 2: Collapsible Subtrees: (a) action b is the best across the information set (b) singleton information set (c) single available action

global improvement, we use $\delta$s as heuristics to decide which parts of a strategy profile are more important to improve. Constraints with high $\delta$s 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 $\delta$ first. Empirical observations indicate that this heuristic is indeed useful; improving constraints with high $\delta$s 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 with the maximum $\delta$, we calculate the required change in $v$ to bring $\delta$ down to 0, assuming all other variables are unchanged. We assess the impact of each potential change on the $\delta$s 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 $\delta_c$. Instead of changing the probabilities of actions involved in the constraint $E(a) - E(b)$ as done above, we can alternatively switch the player's preference for $a$ and $b$ by switching their probabilities. This way, a situation where $\sigma(a,h) = 0$, $\sigma(b,h) = 1$ and $E(a) - E(b) = 0.5$ is rectified by switching the probabilities of $a$ and $b$, thereby satisfying the constraint. We generate points from these 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 $\delta$s 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 $\delta$ 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; the gains are at least as large as the largest $\delta$, but can be much larger. 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 $\epsilon$-BNE, we consider a profile stable if no player stands to make more than $\epsilon$% more reward by deviating from (or completely overhauling) its prescribed strategy. We define a global measure called *Maximum Overall Motivation*(MOM) to deviate. MOM($\sigma$) is the maximum, over all players, upper bound on motivation to deviate from $\sigma$ assuming strategies of other players are held constant. MOM is therefore an upper bound on $\epsilon$. The lower the MOM, the more stable $\sigma$ 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 $\epsilon$%. 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 $\sigma$, we build a modified game $\Gamma_{revealed}$ from the original game $\Gamma$. $\Gamma_{revealed}$ 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 $\Gamma$, 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 ($\delta$) 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.

Table 1 shows three kinds of payoffs achieved by two players playing a strategy profile $\sigma$ in an example game and how they are used to calculate MOM($\sigma$). Payoff($\sigma$) is the payoff from playing $\sigma$ in the original game $\Gamma$. PBR is a player's payoff from its best response strategy; the best he can get in $\Gamma$ given that his opponent's strategy is fixed. It is obtained by calculating the player's payoff in a transformed game where opponent nodes are changed to chance nodes with action probabilities as dictated by $\sigma$. PPI is the payoff when playing the perfect information game $\Gamma_{revealed}$. Note that this quantity is easier to calculate than PBR, which is why we use it in calculating MOM. Since PBR cannot exceed PPI, the overall motivation calculated using PPI is an upper bound on a player's percentage regret. MOM is just the maximum overall motivation over all players. It is easy to see that the following relation holds:

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

In other words, a player's payoff from playing $\sigma$ cannot exceed that from playing its best response which in turn cannot exceed its payoff when all information sets are revealed. In the example shown in the table, player 1 would not benefit from having its information sets revealed (its PBR=PPI). This means that even player 2's disclosing its type would not help player 1 get higher reward.

## 6 Experimental Results

In this section we provide experimental results for two aspects of our algorithm. First, we examine the results of collapsing game trees. We then demonstrate the performance of our anytime search algorithm on collapsed game trees. We consider trees from derived from instances of the view maintenance problem as well as general trees. We conclude with important remarks on the presented results.

Table 1: Calculating Maximum Overall Motivation

| Quantity | Player 1 | Player 2 |
|---|---|---|
| Payoff($\sigma$) | 8.97 | 7.88 |
| Payoff of B.R.(PBR) | 10.39 | 8.48 |
| Payoff in $\Gamma_{revealed}$ (PPI) | 10.39 | 9.09 |
| Overall Motivation (PPI-Payoff($\sigma$))/PPI * 100% | 15.85% | 15.26% |
| MOM | 15.85% | |

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 |

## 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 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. With imperfect 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).

## 6.2 Performance of the Anytime Search Algorithm

Our second set of experiments investigates the performance of our anytime algorithm on VM trees as well as general trees. 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. Table 4 shows the results binned by tree size and amount of time taken to reach a regret of 5% or less. The table shows the percentage of trees in each size bin for which the algorithm could reach the desired regret within the indicated time range. As can be seen, for most of the trees in any given bin, the algorithm demonstrates good anytime behavior by reaching the required level of regret within 100s.

General trees proved to be more challenging. Table 5 shows the time taken to reach a regret of 5% or less

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 |

Table 4: Anytime performance on VM trees (threshold=5%)

| Tree Size | $\leq$ 20 secs | 21-100 secs | 101-500 secs | 501-1000 secs | > 1000 secs |
|---|---|---|---|---|---|
| 0-200 | 100% | - | - | - | - |
| 200-400 | 94% | 2% | 2% | - | 2% |
| 400-1000 | 45.8% | 41.7% | 8.3% | 4.2% | - |
| 1000-2000 | 19% | 50% | 4.2% | - | 16.7% |
| 2000-3600 | - | 63% | 18.5% | - | 18.5% |

(we omitted time bin 500-1000 seconds because it is empty) for 73 random trees, each solved 3 times starting from random points. 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 if we set the desired regret to be 10%, as shown in Table 6. Relaxing the threshold gets us a solution within 500 seconds in at least 75% of the cases. We plan to investigate what characteristics of trees derived from the view maintenance problem make them more amenable to an algorithm like ours.

## Remarks

Some remarks about our results are in order:

- It should be noted that there are many possibilities for fine-tuning 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.

- 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.

Table 5: Anytime performance on general trees (threshold=5%)

| Tree Size | Num. of Trees | $\leq$ 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 6: Anytime performance on general trees (threshold=10%)

| Tree Size | Num. of Trees | $\leq 20$ secs | 21-100 secs | 101-500 secs | $> 2000$ secs |
|---|---|---|---|---|---|
| 0-200 | 105 | 88.6% | - | - | 11.4% |
| 200-400 | 75 | 90.7% | 5.3% | - | 4% |
| 400-600 | 9 | 88.9% | 11.1% | - | - |
| 800-1100 | 30 | 6.7% | 30% | 40% | 23.3% |

- Finally, we would like to note that for most of the games reported in this work, Gambit [26], a library of tools and algorithms for the construction and analysis of games, was unable to find a solution. We omit a detailed comparison because Gambit only returns exact, rather than approximate, equilibria, so we could not assess its progress during execution.

## 7 Related Work

In this section we survey work done in the areas our work relates to; view maintenance, collaboration among self-interested database systems and game theory. Wherever possible, we highlight the differences between our approach and those taken in the related efforts we discuss.

### 7.1 View Invalidation and Maintenance

The view invalidation problem - deciding whether a set of database updates invalidates a view - has been the subject of many studies ([5, 20, 23, 7]). The approach in [7] compares templates characterizing database updates and the view query to decide whether the updates are potentially relevant, in which case the templates are instantiated to check for actual relevance. Polling queries are used in [5] to decide whether the view is invalid. The invalidation module incrementally polls the $DBM$s and determines each poll query based on results of previous polls until a decision regarding the freshness of the view is reached. The tradeoff involved is that between the amount of polling and the decision quality (precise, over- and under-invalidation).

Unlike our setting, most of the work done in view invalidation assumes *DBMs* are cooperative and is therefore not concerned about privacy. One exception is [23] where the tradeoff between privacy and scalability (which is affected by unnecessary invalidations) is considered. Clues are sent, along with encrypted queries and updates, to help the entity responsible for caching results determine whether the cache was invalidated. A *DBM* decides offline what kind of clues should be sent based on the update and view templates. Two schemes for calculating clues are considered. One scheme calculates the clues necessary to guarantee precise invalidations while the other generates clues only when the invalidation savings exceeds the cost of generating the clues. The setting involves a single database manager who reveals all necessary clues in one go.

Our setting differs from the above in having multiple managers who reveal information incrementally. Also, unlike the work in [5] and [23], we reason about the tradeoff between the amount of polling/clues/privacy loss and how up-to-date the view is in a game-theoretic way.

The view maintenance problem is concerned with keeping the view up-to-date when the underlying databases change. Since it is often wasteful to recompute the view from scratch, several approaches have been suggested to implement *incremental* view maintenance where algorithms compute *changes* to the view in response to changes in the underlying databases (e.g. [13, 3]).

There has been much work on preserving privacy when answering queries about a database. The query-view security problem considers the amount of privacy lost, if any, in publishing a given view of a database [27]. From this work, we borrow the notions that different pieces of information have different privacy costs (e.g. a database manager may be unwilling to reveal an employee's salary but willing to reveal his department) and that this cost may depend on the information revealed so far. Sometimes the total loss is super-additive (e.g. when the other party can infer a third piece of information from 2 revealed pieces), while at others it is sub-additive (in the extreme case, the new piece of information can be inferred from previously disclosed information). For simplicity, rather than calculate these privacy costs, we consider them part of the problem specification.

## 7.2 Untrusted, Distributed Database Systems

A database view on a number of base relations is usually defined by a query involving selections and joins over a subset of attributes of these relations. The fact that the *DBM*s in our setting belong to different institutions raises concerns similar to those encountered in secure distributed databases, a line of work which has become increasingly relevant with the increase in database outsourcing. One concern in outsourcing and distributing a database to multiple servers is allowing normal execution of queries without sacrificing privacy. Some approaches to achieving this balance encrypt data before storing it in the (untrusted) external database. To avoid the overhead of encryption and decryption, [1] suggests an approach where the client partitions its data across logically independent database systems (that cannot communicate with each other) in a way that ensures that the exposure of the contents of any one database does not result in a violation of privacy. Privacy in this work is defined as a set of constraints where each constraint is a set of attributes that an adversary should not be able to infer in its entirety on gaining access to a single database. A means of breaking down a query into sub-queries to be executed at the different databases is given, as well as a way of piecing together the results from these databases.

The difference between the above situation and our view maintenance problem is that in the former, privacy is the concern of an a single external entity that is trying to achieve it using distribution. In our problem, privacy is the concern of the database managers themselves. We do not have one entity vs. the adversary. Rather, we have multiple entities who all have privacy concerns when dealing with each other.

The problem of executing a query over distributed database systems that do not trust each other is an example of a more general problem known as *Secure MultiParty Computation* (SMC) [35]. In a multi-party computation (MPC), each participant has private data and the participants want to compute the value of a public function over their variables. A MPC protocol is secure if no participant can learn more from the description of the public function and the result of the global calculation than what he can learn from his own variables. In the distributed database case, the participants are the different database systems and the computation is a query to be executed over the various relations in the systems.

There has been work that focuses on providing SMC protocols for specific kinds of computations. [10] considers the problem of computing the intersection of private datasets of two parties, where the datasets contain lists of elements taken from a large domain. [21] presents protocols for distributed computation of relational intersections and equi-joins such that each site gains no information about the tuples that do not intersect or join with its own. Other work, for example [2], addresses the broader problem of privacy-preserving data mining.

The difference between the SMC problem and ours is that in SMC, the parties follow a protocol to carry out a computation to completion under conditions of security. In our problem, each party is a *decision maker* who can accept or refuse to contribute to different parts of the computation based on his costs and benefits of doing so. Therefore in our case, the multi-party computation does not necessarily proceed to completion. Rather, it only proceeds as long as it is worth it for the involved parties. So our problem is more a decision-making problem than a protocol design one.

Another kind of cost that is often discussed in distributed databases is the communication cost incurred in moving data to the site where they will be operated on. In [8], the authors address the problem of communication-efficient implementation of the SQL *join* operator in a *sensor database*; a sensor network viewed as a distributed database system where each sensor generates a stream of tuples. The distributed implementation of SQL queries in these systems must minimize communication cost, otherwise communicating the enormous amount of data present in a typical sensor network would pose a serious burden on the sensors' limited batteries. The key to reducing communication is solving the *operator placement* problem [33]; finding the best point in the network at which to perform specific processing in order to minimize communication without over-burdening lower-capability devices.

Table 7: Related Work in Game Theory

| | Complete Information | Incomplete Information |
|---|---|---|
| Tree | Authors: Kearns, Littman, Singh<br>Venue: NIPS'01[17], UAI'01[16]<br>Solution: Approx. Exact for limited games<br>Approach: Constraint satisfaction by sweeps up&down tree | Authors: Singh, Soni, Wellman<br>Venue: EC-04[31]<br>Solution: Approx.<br>Approach: Constraint satisfaction by sweeps up&down tree |
| General Graph | Authors:Vickery, Koller<br>Venue: AAAI'02[34]<br>Solution: Approx.<br>Approach:Hill climbing, Constraint satisfaction, Cost minimization | Authors: Soni, Singh, Wellman<br>Venue: AAMAS'07[32]<br>Solution: Approx.<br>Approach: CSP algorithms-like node and arc consistency |

## 7.3 Games with Incomplete Information

Most of the game-theoretic literature gives examples of fairly simple games with incomplete information where a leader agent moves first according to its type and the other player, whose move depends on the observed action of the leader and the follower's own type, follows. Some work discusses games that repeat at every stage, thus making it easier for agents to maintain and update beliefs about others' types and act accordingly. A large fraction of work, however, is descriptive in nature; it describes what constitutes an equilibrium strategy profile, for instance, but does not detail how to find it. Until recently, there has been a dearth of work that deals with computational aspects of finding equilibria efficiently, especially in games with imperfect information. Koller and Pfeffer [19] voice a similar concern and mention some of the consequences of this slow advance in computational game theory. Theirs is one of 2 major works addressing computationally solving imperfect information games; an end-to-end system called Gala which accepts a game description in a human-readable, intuitive format, analyzes it and outputs strategy profiles that make rational sense from a game-theoretic point of view. The other major contribution in this field is that by McKelvey and McLennan [26, 25]; Gambit is a game-theoretic software package with tools and algorithms for the construction and analysis of different types of games.

There have been several attempts in the past to efficiently find approximate equilibria. Most of these attempts address 1-stage games. We discuss some attempts that are most related to our approach and note why they are inadequate for the sequential games we consider. Table 7 gives pointers to some of the works that try to find approximate (and sometimes exact) solutions efficiently by exploiting the structure in agent interaction where possible. The argument is that even in games with several players, there is a locality of interaction that can be captured by a neighborhood graph showing which players have payoffs that are affected by which players' strategies. These *graphical games* have the advantage that if the neighborhood graph is sparse, compact payoff functions can be defined for the game, resulting in representational savings. More importantly, computational savings can be obtained when calculating (approximate) equilibria. The main question is, therefore, whether a given algorithm scales well with respect to the number of players, rather than the number of strategies per player.

The 4 efforts shown in Table 7 address the 4 combinations of games with complete/incomplete information whose neighborhood graph is a tree/general graph.

### 1. Tree Games with Complete Information

The work by Kearns, Littman and Singh on games with complete information and acyclic graphs (henceforth referred to as the KLS algorithm) can be said to have ushered in approaches for the other 3 types of games. The main idea is to have a pair of sweeps up and down the tree which together effect something close to variable elimination in a Constraint Satisfaction Problem (CSP) generated by the graphical game. In the upward sweep from leaves to root, each node $x$ passes to its parent $y$ a table $D(\mu_x, \mu_y)$ indexed by the strategies of $x$ and $y$. If $x$ is a leaf node, $D(\mu_x, \mu_y) = 1$ if $\mu_x$ is a best response to $\mu_y$. If $x$ is an internal node, $D(\mu_x, \mu_y) = 1$ if $\mu_x$ is a best response to $\mu_y$ and to some strategy profile of $x$'s children. At the root, a strategy for the root is chosen based on the tables passed up to it by its children, and a downward sweep begins, in which each parent node dictates a strategy for each of its children based on what its own parent has dictated.

The discretization technique presented in the KLS paper was carried over to all the other papers in our table. Because the space of mixed strategies is continuous, it is necessary to discretize the strategy space of each player to be able to apply constraint satisfaction-like algorithms. The discretization is simple and depends on restricting the probabilities assigned to an action by a strategy to be multiples of some factor $\tau$. The granularity of the discretization affects how close we can get to an exact equilibrium.

Comparing KLS algorithm to ours, we see that KLS provides an exact solution (it solves a CSP rather than a COP) to an approximation of the original problem (obtained through the discretization technique). Our algorithm provides an approximate solution ($\epsilon$-BNE) to the original problem. In fact, even if we discretized the type spaces of the players in our games, it would still be intractable to use backup-based CSP algorithms for variables with finite domains. The main problem is that in 1-shot games, the CSP has a variable for each player $i$ denoting $i$'s strategy; a probability distribution, for each of $i$'s types, over the actions available at the one and only decision point that $i$ faces when it is of a certain type. 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. Alternatively, consider a formulation of the CSP derived from a sequential game with a variable per information set. A variable's domain is the set of all probability distributions over actions at the corresponding information set. In our games, the sequential nature of the decision-making process implies that a decision made at one point affects what should be done in the future. This results in a CSP that has multiple, densely connected variables per player in addition to the interactions among players.

## 2. Tree Games with Incomplete Information

The KLS algorithm was extended by Singh, Soni and Wellman[31] to find approximate BNEs in tree games with incomplete information. The extension is trivial for the case of discrete type spaces; each type is mapped to a discretized unconditional strategy. The number of strategies for an agent is exponential in the size of type space and the number of actions available to it. For continuous types, the paper is restricted to games with a certain class of payoff functions.

## 3. Arbitrary Graphical Games with Complete Information

The work of Vickery and Koller [34] is perhaps the closest to ours in terms of how the space of strategy profiles is searched. Their hill-climbing algorithm does function minimization where the function is 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, an important difference is that because theirs is a 1-stage game, their strategy space for a player is its set of actions, rather than sequences of actions as in our case.

In addition to hill-climbing, Vickery and Koller propose a cost minimization algorithm that operates on games with arbitrary neighborhood graphs. Their third and most interesting contribution is highlighting the analogy between graphical games and Bayesian networks and using subgame decomposition techniques inspired by the clique tree algorithm for Bayesian network inference. Each subgame is solved using hill-climbing and the results combined using a CSP approach.

La Mura and Pearson [29] also address games with complete information. However, they use simulated annealing and do not exploit locality of interactions among players. The decision of whether to accept a new point during simulated annealing is made in a decentralized way, thus eliminating the requirement that payoffs be common knowledge.

## 4. Arbitrary Graphical Games with Incomplete Information

Soni, Singh and Wellman propose the PureProp algorithm for games with incomplete information and arbitrary graphs [32]. Again, a CSP is derived from a game, but here the domain of a variable corresponding to a player is the space of strategy profiles of the player's neighbors in the game graph, including itself. The motivation for this formulation is having binary constraints, as opposed to the work in [34] which has non-binary constraints. PureProp operates on the constraints in a way familiar to CSPs; node and arc consistency followed by backtracking search over the reduced variables domains.

The main advantages of the family of algorithms surveyed above is that they are decentralized. There is no central calculating entity that has access to all payoff information. Instead, localized message passing is enough to reach an (approximate) equilibrium. This decentralized calculation allows scaling w.r.t. the number of players. However, they are all limited to 1-shot games. But just as game structure can be exploited to better solve 1-shot games, it can be exploited to perform the key computational step of algorithms originally due to Govindan and Wilson [12], making them tractable for finding exact equilibria in graphical games [4] in a way that scales with the number of players.

One of the few works concerned with large sequential games of incomplete information is that by Gilpin and Sandholm [11]. 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.

In addition to work like the above that tries to solve abstract games in a domain-independent manner, some work starts from a certain domain and formulates the problem as a game. However, instead of solving the game using generic game-theoretic ways, knowledge about the domain is leveraged to capitalize on whatever special features there are. For example, the work in [9] analyzes bilateral multi-issue negotiation between self-interested agents. The situation can be thought of as a game where agents decide how to allocate the issues between them so as to maximize their individual utilities. This game is solved in way that is strongly dependent on the particular type of problems being solved. This has the advantage of exploiting special features of the domain, but suffers from being very limited in applicability. A similar approach is taken in [18] where the setting has a large number of players who must make individual investment decisions related to security (e.g. physical, financial), but in which the ultimate safety of each participant may depend in a complex way on the actions of the entire population. Again, the situation is formulated as a game but the way and equilibrium is found is specific to their setting.

In this work, we approached the game without drawing on specific features of the View Maintenance problem, but in future work, we plan to investigate what benefits can be obtained from doing so.

## 8 Conclusion and Future Work

The view maintenance problem concerns how a view is refreshed when its underlying data sources are updated. This problem has been extensively studied in settings where view refreshing is expensive. We investigate the view maintenance problem when *self-interested* database managers from different institutions are involved, each concerned about the privacy of its database. 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. 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.

This work has two main contributions. First, we formulate the view maintenance problem as a sequential game of incomplete information. Second, we propose a general anytime algorithm for approximately solving general games of incomplete information by searching the space of strategy profiles for an $\epsilon$-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. Experiments were conducted 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. For view maintenance trees, our anytime search algorithm demonstrates good anytime behavior by reaching a regret level of 5% within 100 seconds for most trees. General game trees are more challenging for the 5% regret level, but the results are much better for the 10% level. Depending on the application, the regret level can be adjusted to the maximum tolerable regret.

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 reduced the size of game trees, especially those derived from the view maintenance problem. A fairly simple search algorithm was able to 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. A related future direction is to devise a domain-specific algorithm that capitalizes on these special features to solve view maintenance trees more efficiently. A third 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.

## References

[1] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *Proceedings of the Second Biennial Conference on Innovative Data Systems Research*, CA, USA, 2005.

[2] R. Agrawal and R. Srikant. Privacy preserving data mining. In *Proceedings of the ACM SIGMOD*, 2000.

[3] 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.

[4] B. Blum, C. R. Shelton, and D. Koller. A continuation method for nash equilibria in structured games. *Journal of Artificial Intelligence Research*, 2006.

[5] 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.

[6] 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.

[7] 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.

[8] V. Chowdhary and H. Gupta. Communication-efficient implementation of join in sensor networks. In *Proceedings of DASFAA*, Beijing, China, 2005.

[9] 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.

[10] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proceedings of Advances in Cryptology, EUROCRYPT*, 2004.

[11] 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.

[12] S. Govindan and R. Wilson. Structure theorems for game trees. In *Proceedings of the National Academy of Sciences*, 2002.

[13] 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.

[14] P. Haas and A. Swami. Sequential sampling procedures for query size estimation. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, 1992.

[15] J. C. Harsanyi. Games with incomplete information played by bayesian players, part i. the basic model. *Management Science*, pages 159–182, November 1967.

[16] M. Kearns, M. Littman, and S. Singh. An efficient exact algorithm for singly connected graphical games. In *Proceedings of Advances in Neural Information Processing Systems*, British Columbia, Canada, 2001.

[17] 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.

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

[19] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, pages 167–215, 1997.

[20] A. Y. Levy and Y. Sagiv. Queries independent of updates. In *Proceedings of the 19th Very Large Data Bases Conference*, Dublin, Ireland, 1993.

[21] 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.

[22] C. A. Lynch. Selectivity estimation and query optimization in large databases with highly skewed distribution of column values. In *Proceedings of the Fourteenth International Conference on Very Large Data Bases*, California, USA, 1998.

[23] 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.

[24] 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.

[25] R. McKelvey and A. McLennan. Computation of equilibria in finite games. *Handbook of Computational Economics*, pages 87–142, 1996.

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

[27] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. *Journal of Computer and System Sciences (JCSS)*, 2006.

[28] R. B. Myerson. *Game Theory: Analysis of Conflict.* Harvard University Press, 1991.

[29] 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.

[30] Y. Shoham, R. Powers, and T. Grenager. Multi-agent reinforcement learning: A critical survey. Technical report, Stanford University, 2003.

[31] 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.

[32] 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.

[33] U. Srivastava, K. Munagala, and J. Widom. Operator placement for innetwork stream query processing. In *Proceedings of PODS 2005 (ACM Symposium on Principles of Databases)*, Maryland, USA, 2005.

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

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