MayBMS: A Probabilistic Database Management System

Jiewen Huang[1,2], Lyublena Antova[2], Christoph Koch[2], and Dan Olteanu[1]

[1]Computing Laboratory, Oxford University, Oxford, OX1 3QD, UK
{jiewen.huang,dan.olteanu}@comlab.ox.ac.uk
[2]Department of Computer Science, Cornell University, Ithaca, NY 14853, USA
{lantova,koch}@cs.cornell.edu

ABSTRACT

MayBMS is a state-of-the-art probabilistic database management system which leverages the strengths of previous database research for achieving scalability. As a proof of concept for its ease of use, we have built on top of MayBMS a Web-based application that offers NBA-related information based on what-if analysis of team dynamics using data available at www.nba.com.

Categories and Subject Descriptors: H.2.4 [Database Management]: Systems – *Query Processing*

General Terms: Algorithms, Languages, Performance

Keywords: Query Processing, Probabilistic Databases

1. INTRODUCTION

Database systems for uncertain and probabilistic data promise to have many applications. Query processing on uncertain data occurs in the contexts of data warehousing, data integration, and Web data extraction. Data cleaning can be fruitfully approached as a problem of taming uncertainty in the data. Decision support and diagnosis systems employ hypothetical (what-if) queries. Scientific databases, which store outcomes of scientific experiments, frequently contain uncertain data such as incomplete observations or imprecise measurements. Sensor and RFID data are inherently uncertain. Applications in the contexts of fighting crime, tracking moving objects, surveillance, and plagiarism detection essentially rely on techniques for processing and managing large uncertain datasets. Beyond that, many further potential applications of probabilistic databases exist and will manifest themselves once such systems become available.

The *MayBMS* system (note: MayBMS is read as "maybe-MS", like DBMS) is a complete probabilistic database management system that leverages robust relational database technology. The MayBMS system has been under development since 2005 and has undergone several transformations. MayBMS has been released and is available for download at

http://maybms.sourceforge.net.

A fundamental design choice that sets MayBMS apart from existing research prototypes such as Trio and MystiQ is that MayBMS is an extension of the open-source PostgreSQL *server backend*, and not a front-end application of PostgreSQL. Our backend is easily accessible through multiple APIs (inherited from PostgreSQL), and has efficient internal operators for processing probabilistic data.

Several demonstration scenarios (data cleaning using constraints, human resources management, and analysis of social networks) are available at the MayBMS website. To demonstrate how easily applications can be built on top of MayBMS, we used PHP to construct a website that offers NBA[1]-related information based on what-if analysis of team dynamics (player fitness, skill management) using data available at www.nba.com. We show for instance how such an application can predict the players' fitness by simulating random walks on stochastic matrices encoding the transitions in player's fitness based on severity of recent injuries.

SIGMOD'09, June 29–July 2, 2009, Providence, Rhode Island, USA.
ACM 978-1-60558-551-2/09/06.

2. SYSTEM OVERVIEW

MayBMS stores probabilistic data in U-relational databases, a succinct and complete representation system for large sets of possible worlds [1]. Queries are expressed in an extension of SQL with specialized constructs for probability computation and what-if analysis [4]. In addition, MayBMS uses several state-of-the-art exact and approximate confidence computation techniques [2, 3, 5].

2.1 U-Relational Databases

A U-relational database consists of a set of U-relations. U-relations are standard relations extended with condition and probability columns to encode correlations between the uncertain values and probability distribution for the set of possible worlds [1]. The condition columns store variables from a finite set of independent random variables and their assignments; the probability columns store the probabilities of the variable assignments occurring in the same tuple. A U-relation can have several such condition (and probability) columns. Attribute-level uncertainty is achieved through vertical decompositions, and an additional (system) column is used for storing tuple ids and undoing the vertical decomposition on demand.

2.2 The MayBMS Query Language

The MayBMS query language extends SQL with uncertainty-aware constructs [4]. Its features include genericity, compositionality, and a well-understood relationship to existing query languages. An important subset of its constructs is presented next.

[1]NBA stands for National Basketball Association.

In the sequel, U-relations without condition and probability columns, which correspond to standard relations, are called *typed-certain (t-certain) tables.* The MayBMS query language has constructs that map (i) uncertain tables to t-certain tables, such as confidence computation constructs, (ii) uncertain to uncertain and t-certain to t-certain tables, such as full SQL, and (iii) t-certain to uncertain tables, such as constructs that extend the hypothesis space and create new possible worlds. Some restrictions are in place to assure that query evaluation is feasible. In particular, we do not support the standard SQL aggregates such as `sum` or `count` on uncertain relations (but we do support expectations of aggregates). This can be easily justified: in general, these aggregates will produce exponentially many different numerical results in the various possible worlds, and there is no way of representing exactly these results in an efficient manner. In addition to standard SQL, MayBMS supports the following uncertainty-aware constructs:

1. conf, aconf, tconf, and possible: These constructs map uncertain tables to t-certain tables consisting of tuples possible in some of the worlds represented by the input, with or without their exact or approximate confidences.

The construct `conf` returns the exact confidence of each distinct tuple, while aconf(ϵ, δ) computes an (ϵ, δ)-approximation of this confidence, i.e., the probability that the computed value $\hat{p}$ deviates from the correct probability p by more than $\epsilon \cdot p$ is less than δ. Syntactically, the confidence computation constructs `conf` and `aconf` are treated like SQL aggregates. By using aggregation syntax and not supporting `select distinct` on uncertain relations, we avoid the need for conditions beyond the special conjunctions that can be stored with each tuple in U-relations.

The construct `tconf` computes the marginal probability of each tuple in isolation from the other (possibly duplicate) tuples. The construct `possible` can be added to `select` statements and its effect is to filter out the tuples with probability zero and eliminate the duplicates. It can thus be reformulated using `tconf`.

2. repair-key and pick-tuples: These constructs map t-certain tables to uncertain tables. Conceptually, `repair-key` takes a set of attributes $\vec{K}$ and a relation R as arguments and nondeterministically chooses a maximal repair of key $\vec{K}$ in R, that is, it removes a minimal set of tuples from R such that the key constraint is no longer violated. The `repair-key` operation accepts an optional argument that allows us to assign nonuniform probabilities to the possible choices.

The construct `pick-tuples` creates a probabilistic relation representing all the possible subsets of the input table.

Note that repair-key and pick-tuples are queries, rather than update statements. They have the following syntax:

- repair key <attributes> in <t-certain-query>
  [weight by <expression>]
- pick tuples from <t-certain-query>
  [independently] [with probability <expression>]

The parameter [independently] ensures that the output probabilistic relation is tuple-independent.

3. argmax: The aggregate argmax(arg,value) outputs all the `arg` values in a group (specified by the group-by clause) whose tuples have a maximum `value` within that group.

4. esum and ecount: Although the standard SQL aggregates are forbidden on uncertain relations, MayBMS supports aggregate operations on uncertain relations such as `esum` and `ecount`, which compute expected sums and counts across groups of tuples. While it may seem that these aggregates are at least as hard as confidence computation (which is #P-hard), this is in fact not so. These aggregates can be efficiently computed using linearity of expectation.

Uncertain queries can be constructed from t-certain queries (queries that produce t-certain tables), `select-from -where` queries over uncertain tables, the multiset union of uncertain queries (using SQL `union`), and `repair-key` and `pick-tuples` statements. The `select-from-where` queries may use any t-certain subqueries in the conditions, plus uncertain subqueries in IN-conditions that occur positively.

2.3 Query Processing

MayBMS evaluates queries on top of U-relations.

Positive relational algebra: The answers to positive relational algebra queries (without confidences) can be computed using a parsimonious translation of such queries into (again) positive relational algebra queries that are then evaluated in standard relational way on U-relations [1].

Approximate confidence computation: The approximation algorithm used by MayBMS is a combination of the Karp-Luby unbiased estimator for DNF counting in a modified version adapted to confidence computation in probabilistic databases, and the Dagum-Karp-Luby-Ross optimal algorithm for Monte Carlo estimation [2]. The latter is based on sequential analysis and determines the number of invocations of the Karp-Luby estimator needed to achieve the required bound by running the estimator a small number of times to estimate its mean and variance.

Exact confidence computation: Our exact algorithm for confidence computation is described in [3]. Given a DNF (of which each clause is a conjunctive local condition), the algorithm employs a combination of variable elimination and decomposition of the DNF into independent subsets of clauses (i.e., subsets that do not share variables), with cost-estimation heuristics for choosing whether to use the former (and for which variable) or the latter. Outside a narrow range of variable-to-clause count ratios, it outperforms the approximation techniques [3]. For tractable queries on probabilistic databases, MayBMS uses the SPROUT codebase [5] for scalable query processing by reduction of confidence computation to a sequence of SQL-like aggregations.

Updates, concurrency control, and recovery: As a consequence of our choice of a purely relational representation system, these issues cause surprisingly little difficulty. U-relations are represented relationally and updates are just modifications of these tables that can be expressed using the standard SQL update operations.

2.4 Implementation

MayBMS is built entirely inside PostgreSQL. The major changes lie in the system catalog, parser, and executor. U-relations are implemented by storing the variables and their possible assignments as pairs of integers, and probabilities as floating-point numbers. The system catalog can distinguish between U-relations and standard relational tables. Confidence computation and other aggregates such as `esum` and `ecount` are registered in the system catalog and implemented as operators in the PostgreSQL executor. The constructs `repair-key`, `pick-tuples`, and `possible` are implemented by rewriting to SQL.

Fitness stochastic matrix For player Bryant			
	F	SE	SL
F	0.8	0.05	0.15
SE	0.1	0.6	0.3
SL	0.8	0.0	0.2

FT (FitnessTransition)			
Player	Init	Final	P
Bryant	F	F	0.8
Bryant	F	SE	0.05
Bryant	F	SL	0.15
Bryant	SE	F	0.1
Bryant	SE	SE	0.6
Bryant	SE	SL	0.3
Bryant	SL	F	0.8
Bryant	SL	SL	0.2
Other players			

U-relation R2 (1-step random walk on FT)				
Player	Init	Final	condition	P
Bryant	F	F	$x \mapsto 1$	0.8
Bryant	F	SE	$x \mapsto 2$	0.05
Bryant	F	SL	$x \mapsto 3$	0.15
Bryant	SE	F	$y \mapsto 1$	0.1
Bryant	SE	SE	$y \mapsto 2$	0.6
Bryant	SE	SL	$y \mapsto 3$	0.3
Bryant	SL	F	$z \mapsto 1$	0.8
Bryant	SL	SL	$z \mapsto 2$	0.2
Other players				

Figure 1: Random walk on a stochastic matrix.

3. HUMAN RESOURCE MANAGEMENT

We have developed several applications on top of MayBMS. They are available at the MayBMS website. We next discuss an implemented application for risk management in the human resources space, in the context of basketball.

Using PHP, we have built on top of MayBMS a web-based application that offers NBA-related decision support functionality based on what-if analysis of team dynamics such as player fitness and skill management. The application uses data available at www.nba.com.

Team management. In the pre-season period, the manager intends to attract new players to strengthen the team. An important question is whether the skills status of the team can be improved. For this, we compute for each skill (such as defense, three-point, and free shooting) the probability that someone with that skill will be playing in the team given the current status of the players (injured, top-form). In a scenario of financial crisis, the team budget is reduced and the manager intends to lay off some players with high salaries but at the same time without compromising the competitiveness of the team significantly. For instance, we may want to keep the availability of skill shooting at least 90% and of passing at least 95%. The manager needs to know whether this is possible and who can be laid off.

Performance prediction. Coaches would like to predict the performance of players, for example how many points a player will score in the next game. Based on the player's recent performance, one can build a simple model to calculate this: if we associate higher weights to the more recent performance of the players, their predicted performance can be expressed in terms of the weighted points.

Fitness prediction. Suppose there is a must-win match three days later and some of the key players of the team have been recently plagued by injuries. The report from the team doctor shows that the players are likely to get injured and the time for comeback varies depending on the recovery progress. We can model the fitness of each player using a stochastic matrix that states the probabilities for one-day transitions between states such as fit (F), seriously injured (SE), and slightly injured (SL). Asking for the three-day fitness of a player can be performed as a random walk of length three on this matrix. Random walks can be encoded as queries using `repair-key` and confidence computation on top of relational encodings of stochastic matrices. Figure 1 gives a stochastic matrix, its relational encoding FT, and the U-relation R2 representing a 1-step random walk on FT. The 3-step random walk on FT is achieved by the following query statements, where the initial state of each player is considered given in a (certain) table States (Player, State).

```
create table FT2 as
select R1.Player, R1.Init, R2.Final, conf() as p from
(repair key Player, Init in FT weight by p) R1,
(repair key Player, Init in FT weight by p) R2, States S
where R1.Player = S.Player and R1.Init = S.State
and R1.Final = R2.Init and R1.Player = R2.Player
group by R1.Player, R1.Init, R2.Final;

select R1.Player, R2.Final as State, conf() as p from
(repair key Player, Init in FT2 weight by p) R1,
(repair key Player, Init in FT weight by p) R2
where R1.Final = R2.Init and R1.Player = R2.Player
group by R1.player, R2.Final;
```

A 1-step random walk on FT is performed by nondeterministically choosing from each Init state of each player a possible Final state using the repair-key construct. We encode it as a U-relation (see R2) that only adds to FT a condition column over independent random variables x, y, and z, which are used to express the correlations created by repair-key: for each Init value, the possible Final values are mutually exclusive, and the choices of Init values are pairwise independent. A 2-step random walk is expressed as a join of two 1-step walks, whereby the Final state of the first walk becomes the Init state of the second. The probability that each player has a certain state is computed using the conf() construct. The table FT2 encodes the stochastic matrix representing the product of the initial stochastic matrix with itself. For a 3-step random walk, we join the outcome of the previous 2-step walk with a 1-step walk.

Acknowledgements. This material is based upon work supported by the National Science Foundation under grant number IIS-0812272, the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001, and by New York State Science Technology and Academic Research under agreement number C050061. Any opinions, findings, conclusions or recommendations expressed are those of the authors(s) and do not necessarily reflect the views of the funding agencies. Jiewen Huang is supported by a one-year scholarship from Cornell University.

4. REFERENCES

[1] L. Antova, T. Jansen, C. Koch, and D. Olteanu. "Fast and Simple Relational Processing of Uncertain Data". In *Proc. ICDE*, 2008.

[2] P. Dagum, R. M. Karp, M. Luby, and S. M. Ross. "An Optimal Algorithm for Monte Carlo Estimation". *SIAM J. Comput.*, **29**(5):1484–1496, 2000.

[3] C. Koch and D. Olteanu. "Conditioning Probabilistic Databases". In *Proc. VLDB*, 2008.

[4] C. Koch, D. Olteanu, L. Antova, and J. Huang. *MayBMS: A Probabilistic Database System. User Manual.* http://maybms.sourceforge.net/manual/,2009.

[5] D. Olteanu, J. Huang, and C. Koch. "SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases". In *Proc. ICDE*, 2009.